RADIANT & Think Tank - Complete Technical Documentation v4.18.0

- RADIANT & Think Tank Executive Summary
  - What is RADIANT?
  - What is Think Tank?
  - Key Differentiators
    - 1. AGI-Driven Model Selection
    - 2. 49 Proven Orchestration Patterns
    - 3. Enterprise-Grade Security
  - Platform Components
  - By the Numbers
  - Use Cases
    - Enterprise AI Gateway
    - Complex Problem Solving (Think Tank)
    - Quality-Critical Applications
    - Cost Optimization
  - Competitive Advantages
  - Technology Stack
  - Deployment Model
  - Pricing Model
  - Roadmap Highlights
  - Summary
- RADIANT Platform Architecture
  - Table of Contents
  - 1. Platform Overview
    - 1.1 What is RADIANT?
    - 1.2 Core Value Proposition
    - 1.3 Platform Statistics
  - 2. System Architecture
    - 2.1 High-Level Architecture
    - 2.2 Three-Component Structure
  - 3. Component Deep Dive
    - 3.1 Model Router Service
    - 3.2 Service Architecture
  - 4. Data Architecture
    - 4.1 Database Schema Overview
    - 4.2 Multi-Tenant Data Isolation
  - 5. Security Architecture
    - 5.1 Security Layers
  - 6. Deployment Architecture
    - 6.1 AWS Infrastructure
    - 6.2 CDK Stack Dependencies
  - 7. Integration Points

7.1 External API Integrations
Think Tank Platform Architecture
Table of Contents
1. Platform Overview
1.1 What is Think Tank?
1.2 Think Tank vs Traditional Chat
1.3 Key Capabilities
2. Core Architecture
2.1 System Components
2.2 Think Tank Engine
3. Problem Solving Pipeline
3.1 Pipeline Stages
3.2 Step Recording
4. Session Management
4.1 Session Lifecycle
4.2 Session Data Model
5. Collaboration Features
5.1 Real-Time Collaboration
6. Domain Modes
6.1 Specialized Reasoning Modes
7. Quality & Confidence
7.1 Confidence Scoring System
8. User Interface
8.1 Think Tank UI Layout
AGI & Workflow Orchestration
1. Overview
Why 50-300% Improvement?
Key Capabilities
Improvement by Use Case
2. The 49 Orchestration Patterns
Pattern Categories
3. AGI Dynamic Model Selection
How It Works
Domain Detection Keywords
4. Model Execution Modes
5. Parallel Execution
Execution Modes
Synthesis Strategies
6. Visual Workflow Editor
Editor Features
Step Configuration
7. API Usage
Execute Workflow
8. Benefits
RADIANT & Think Tank Complete Features List
Feature Categories
1. AI Model Management
1.1 Model Router Service
1.2 Model Metadata Service
1.3 Supported Models (106+)
2. Orchestration & Workflows
2.1 Orchestration Patterns (49)
2.2 AGI Dynamic Model Selection
2.3 Model Execution Modes (9)

2.4 Parallel Execution
2.5 Visual Workflow Editor
3. Think Tank Platform
3.1 Problem Solving Engine
3.2 Session Management
3.3 Domain Modes (8)
3.4 Collaboration
4. Billing & Cost Management
4.1 Credit System
4.2 Subscriptions
4.3 Cost Management
5. Multi-Tenant Platform
5.1 Tenant Management
5.2 User Management
5.3 API Key Management
6. Security & Compliance
6.1 Data Security
6.2 Authentication
6.3 Compliance
7. Analytics & Monitoring
7.1 Usage Analytics
7.2 Model Performance
7.3 Business Intelligence
8. Developer Tools
8.1 SDK
8.2 Webhooks
8.3 Integrations
9. Admin Dashboard
9.1 Dashboard Pages
9.2 UI Features
10. Swift Deployer App
10.1 Deployment Features
10.2 QA & Testing
10.3 AI Assistant
10.4 Local Storage
RADIANT Services Reference
Complete Lambda Services Inventory (62 Services)
Core Infrastructure Services
AI Model Services
Orchestration Services
Billing Services
Cognitive Services
Memory Services
AGI Services
Collaboration Services
Additional Services (30-62)
RADIANT Database Schema Reference
Complete Migration Inventory (40 Migrations)
Migration Index
Core Tables (Migration 001)
tenants
users
administrators
approval_requests

- Think Tank Tables (Migration 016)
  - thinktank_sessions
  - thinktank_steps
  - thinktank_tools
- Orchestration Tables (Migration 024)
  - workflow_definitions
  - workflow_tasks
  - workflow_executions
  - task_executions
- Billing Tables (Migrations 033-035)
  - credit_balances
  - credit_transactions
  - subscription_tiers
  - subscriptions
- Row-Level Security (Migration 002)
  - RLS Pattern
  - Setting Tenant Context
  - Tables with RLS Enabled:
- Common Patterns
  - Updated At Trigger
  - Soft Delete Pattern
  - Audit Columns

Swift Deployer App Reference

- App Architecture
  - Overview
  - File Structure (36 Files)
- Models
  - Configuration.swift
  - Credentials.swift
  - Deployment.swift
- Services
  - CDKService.swift
  - DeploymentService.swift
  - AIAssistantService.swift
  - LocalStorageManager.swift
  - HealthCheckService.swift
- Components
  - MacOSComponents.swift
  - AppCommands.swift
- UI Patterns (10 macOS Patterns)
- Dashboard Architecture
  - Technology Stack
  - Page Structure
- Complete Page Inventory (43 Pages)
  - Core Administration
  - AI & Models
  - Orchestration
  - Think Tank
  - Billing & Cost
  - Analytics & Monitoring
  - AGI & Learning
  - Collaboration & Features
- Key Pages Detail
  - Overview Dashboard (/)

Models Page (/models)
Orchestration Patterns Page (/orchestration-patterns)
Visual Workflow Editor (/orchestration-patterns/editor)
Billing Page (/billing)
Analytics Page (/analytics)
Security Page (/security)
Think Tank Page (/thinktank)
Component Library
Shared Components
API Integration
API Client (lib/api.ts)
Authentication (lib/auth.ts)

RADIANT & Think Tank Executive Summary

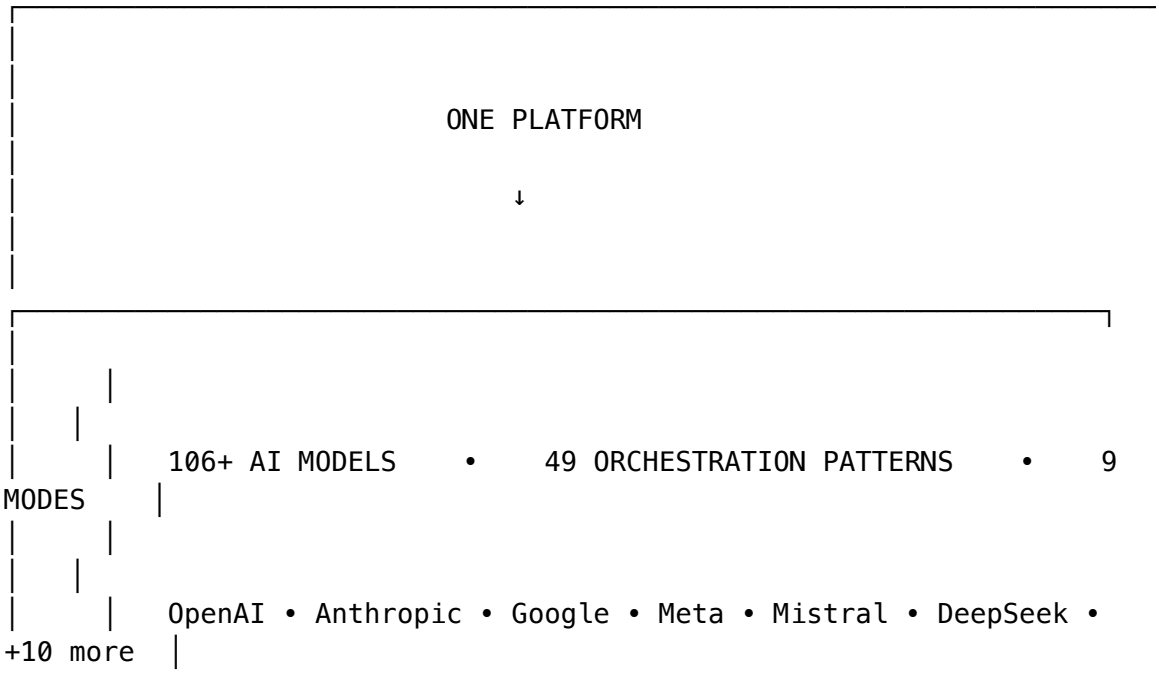
Enterprise AI Platform Overview

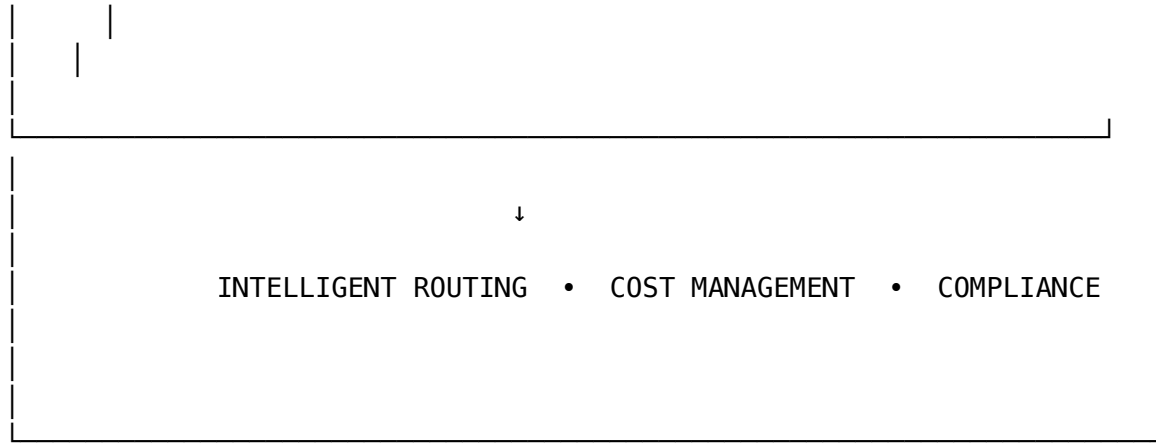
Version 4.18.0 | December 2024

For executives, investors, and decision-makers

What is RADIANT?

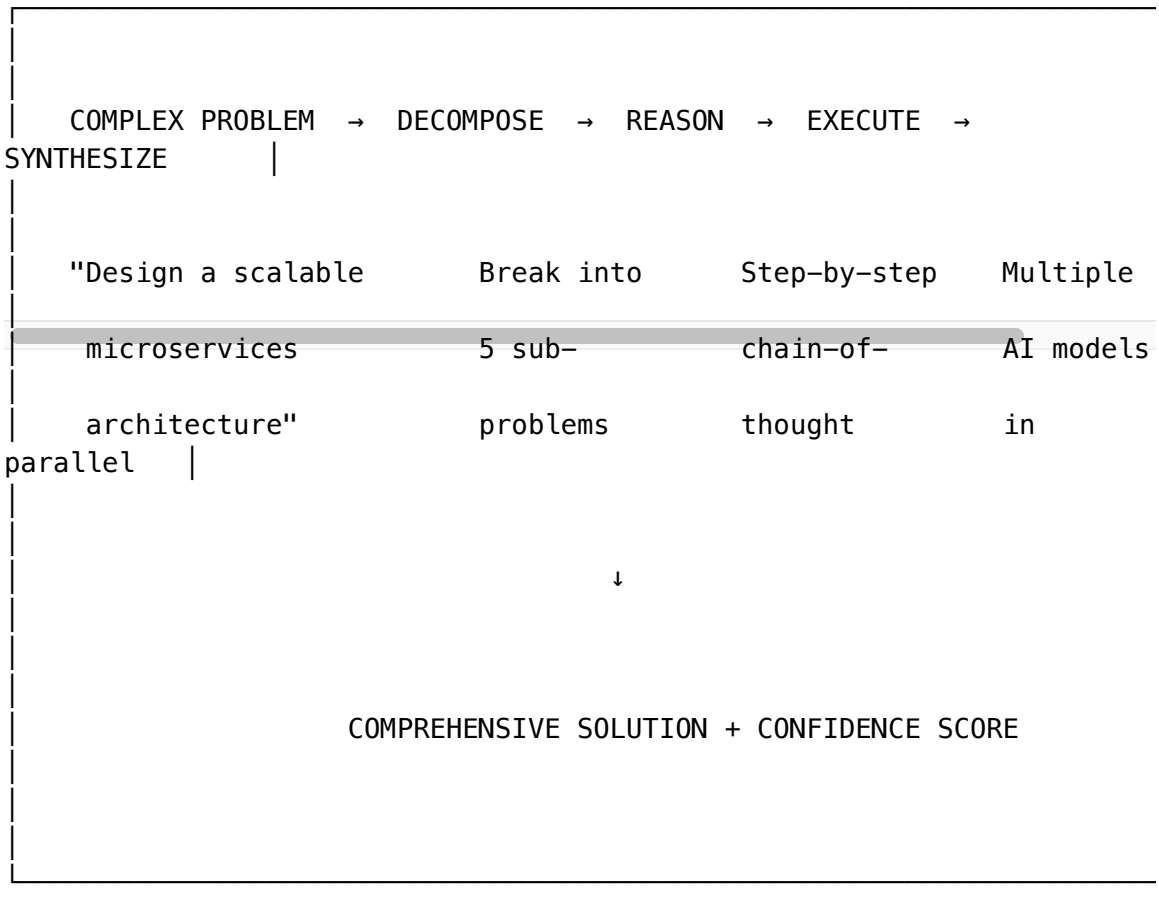
RADIANT is an enterprise-grade, multi-tenant AI platform that provides organizations with unified access to 106+ AI models through a single API, with intelligent orchestration that coordinates multiple AI systems to deliver superior results.





What is Think Tank?

Think Tank is RADIANT's advanced problem-solving platform that decomposes complex problems into manageable steps, applies multi-AI reasoning, and synthesizes comprehensive solutions with confidence scoring.



Key Differentiators

1. AGI-Driven Model Selection

Unlike platforms that use a single AI model, RADIANT's AGI layer **automatically selects the optimal combination of models** based on task analysis:

What We Analyze	What We Select
Problem domain (coding, legal, medical…)	Best models for that domain
Task complexity	Number of models (2-5)
Reasoning requirements	Execution mode (thinking, fast, precise…)
Quality vs speed priority	Parallel execution strategy

Result: 50-300% better outcomes than single-model approaches.

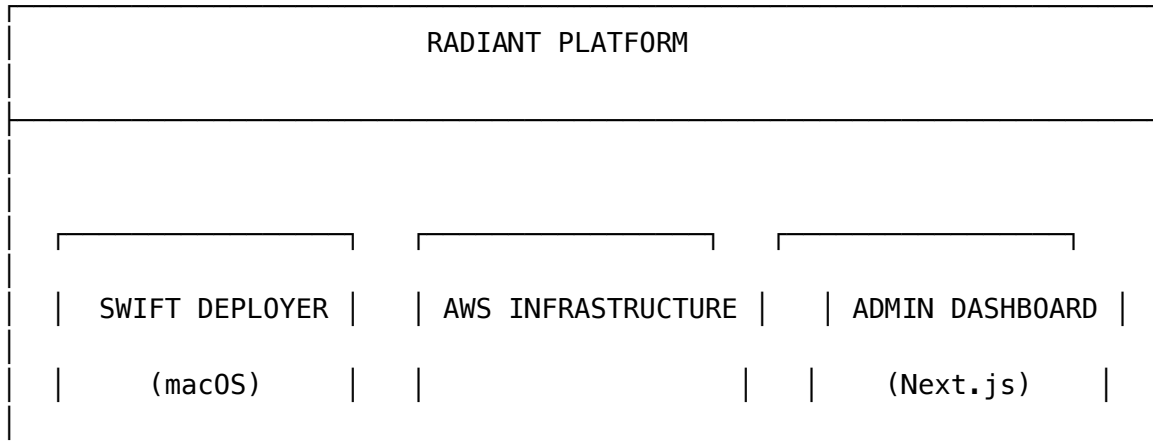
2. 49 Proven Orchestration Patterns

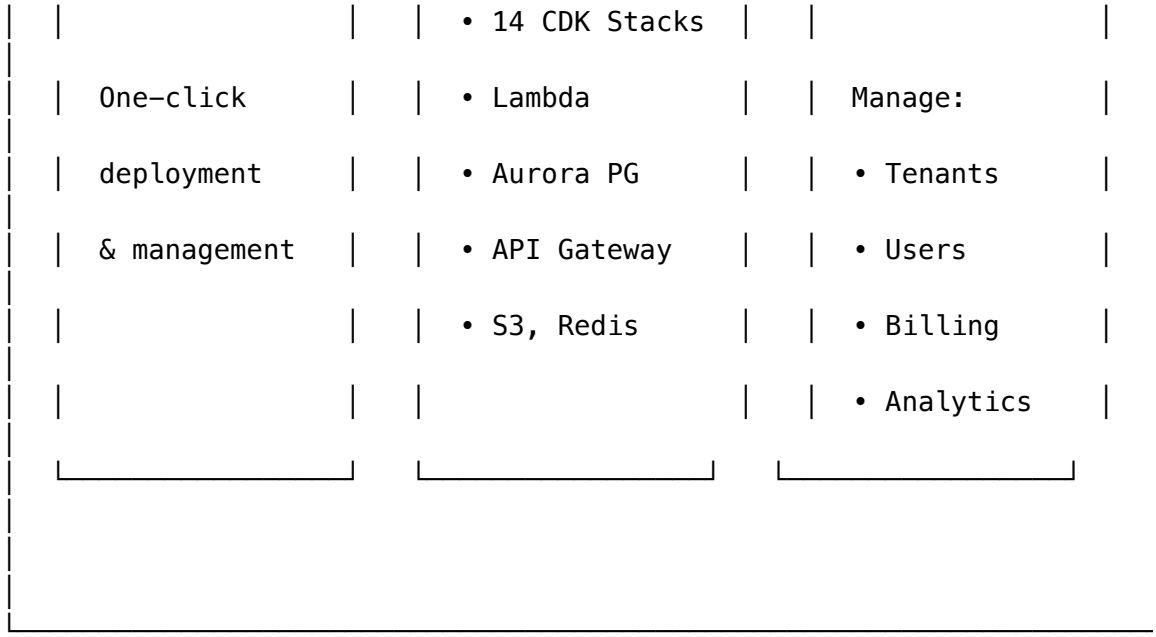
Research-backed workflows including: - **AI Debate** - Two AIs argue, judge decides - **Self-Refine** - Generate → Critique → Improve - **Chain-of-Verification** - Fact-check every claim - **Tree of Thoughts** - Explore multiple solution paths

3. Enterprise-Grade Security

Capability	Description
Multi-Tenant Isolation	PostgreSQL Row-Level Security
Compliance	SOC2, HIPAA-ready
Encryption	At-rest (AES-256) and in-transit (TLS 1.3)
Audit Logging	Complete activity trail

Platform Components





By the Numbers

Metric	Value
AI Models	106+ (50 external + 56 self-hosted)
AI Providers	15+ integrated
Orchestration Patterns	49 documented workflows
Execution Modes	9 specialized modes
Database Migrations	66+ schema versions
CDK Stacks	14 infrastructure components

Use Cases

Enterprise AI Gateway

- Unified access to all major AI providers
- Centralized cost management and budgeting
- Consistent API regardless of backend model
- Automatic failover for reliability

Complex Problem Solving (Think Tank)

- Multi-step technical analysis
- Research synthesis with citations
- Architecture design with artifacts
- Decision support with confidence scores

Quality-Critical Applications

- Legal document analysis (precise mode)
- Medical information processing (HIPAA compliant)
- Financial analysis (multi-model verification)
- Code generation (AI debate + critique)

Cost Optimization

- Intelligent model routing (use cheaper models when appropriate)
- Budget alerts and limits
- Usage analytics by team/project
- Model performance vs cost analysis

Competitive Advantages

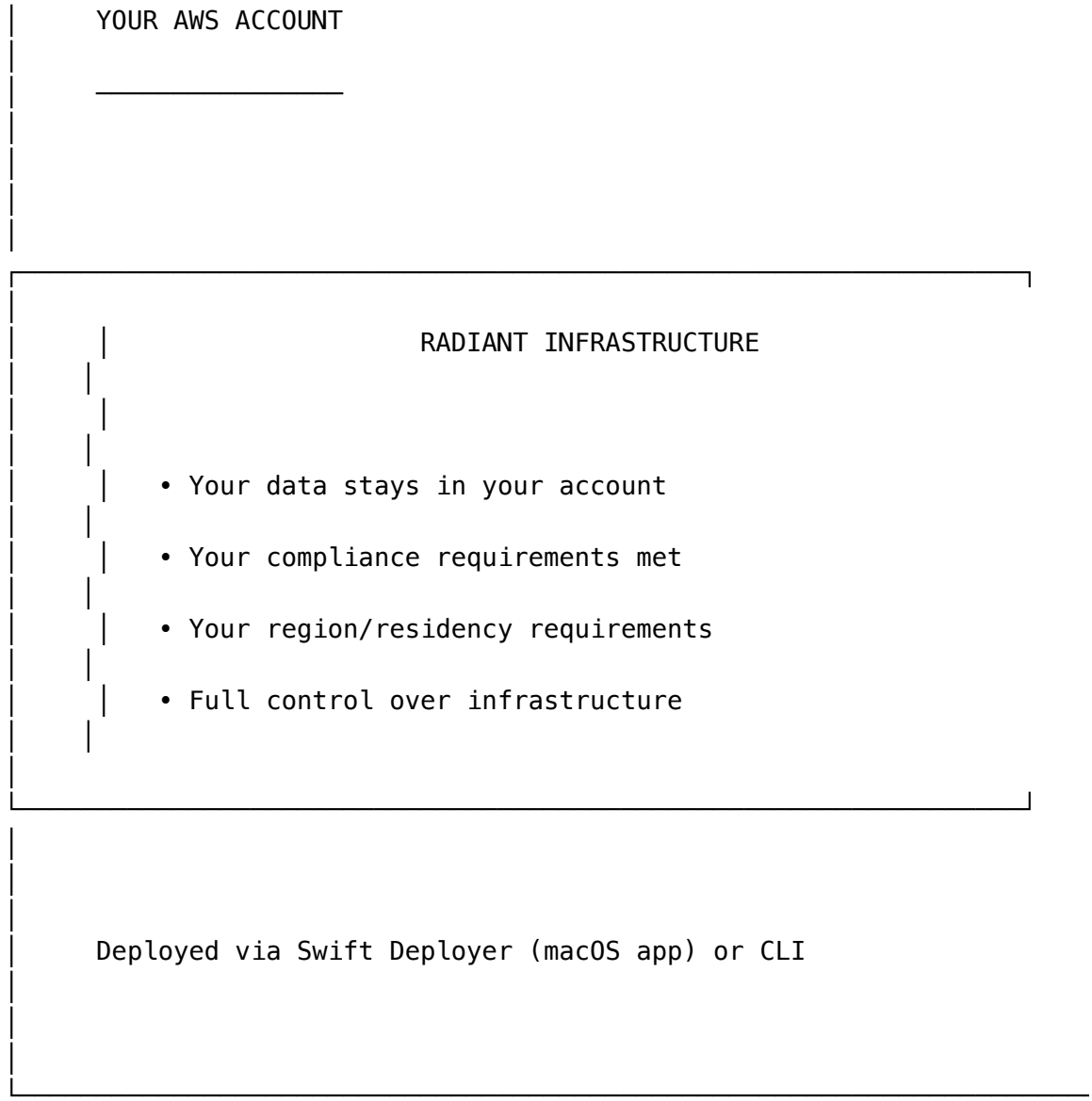
vs. Single-Model APIs	vs. Other Platforms
✓ Multi-model orchestration	✓ 49 research-backed patterns
✓ Built-in verification	✓ AGI-driven model selection
✓ Higher accuracy	✓ 9 execution modes
✓ Reduced bias	✓ Visual workflow editor
✓ Confidence scoring	✓ Think Tank problem solving

Technology Stack

Layer	Technology
Frontend	Next.js 14, TypeScript, Tailwind CSS, shadcn/ui
Backend	AWS Lambda (Node.js 20), API Gateway
Database	Aurora PostgreSQL (Serverless), DynamoDB, Redis
Infrastructure	AWS CDK (TypeScript), 14 stacks
Desktop	SwiftUI (macOS 13.0+, Swift 5.9+)
Security	Cognito, KMS, WAF, Row-Level Security

Deployment Model

RADIANT deploys to **your AWS account**:



Pricing Model

Tier	Target	Includes
Free	Developers	10K tokens/month, 3 models
Pro	Teams	1M tokens/month, all models, orchestration
Enterprise	Organizations	Unlimited, SLA, custom patterns, HIPAA

All tiers include: - Full API access - Admin dashboard - Basic analytics - Email support

---

Roadmap Highlights

Timeframe	Features
Q1 2025	Mobile SDK, more self-hosted models
Q2 2025	Fine-tuning pipeline, custom model hosting
Q3 2025	Multi-region deployment, advanced compliance
Q4 2025	Marketplace for custom patterns

Summary

RADIANT + Think Tank delivers:

1. **Unified AI Access** - One API for 106+ models across 15+ providers
2. **Intelligent Orchestration** - AGI selects optimal models and modes
3. **Superior Results** - 49 patterns achieve 50-300% better outcomes
4. **Enterprise Security** - Multi-tenant, SOC2, HIPAA-ready
5. **Cost Control** - Budgets, analytics, intelligent routing
6. **Problem Solving** - Think Tank for complex multi-step reasoning

---

RADIANT v4.18.0 + Think Tank v3.2.0

The enterprise platform for intelligent AI orchestration

Contact: info@radiant.ai | **Documentation:** docs.radiant.ai

RADIANT Platform Architecture

Enterprise Multi-Tenant AI Platform

Version 4.18.0 | December 2024

A comprehensive technical architecture document for the RADIANT AI orchestration platform

---

Table of Contents

1. Platform Overview
2. System Architecture
3. Component Deep Dive
4. Data Architecture
5. Security Architecture

6. Deployment Architecture
7. Integration Points

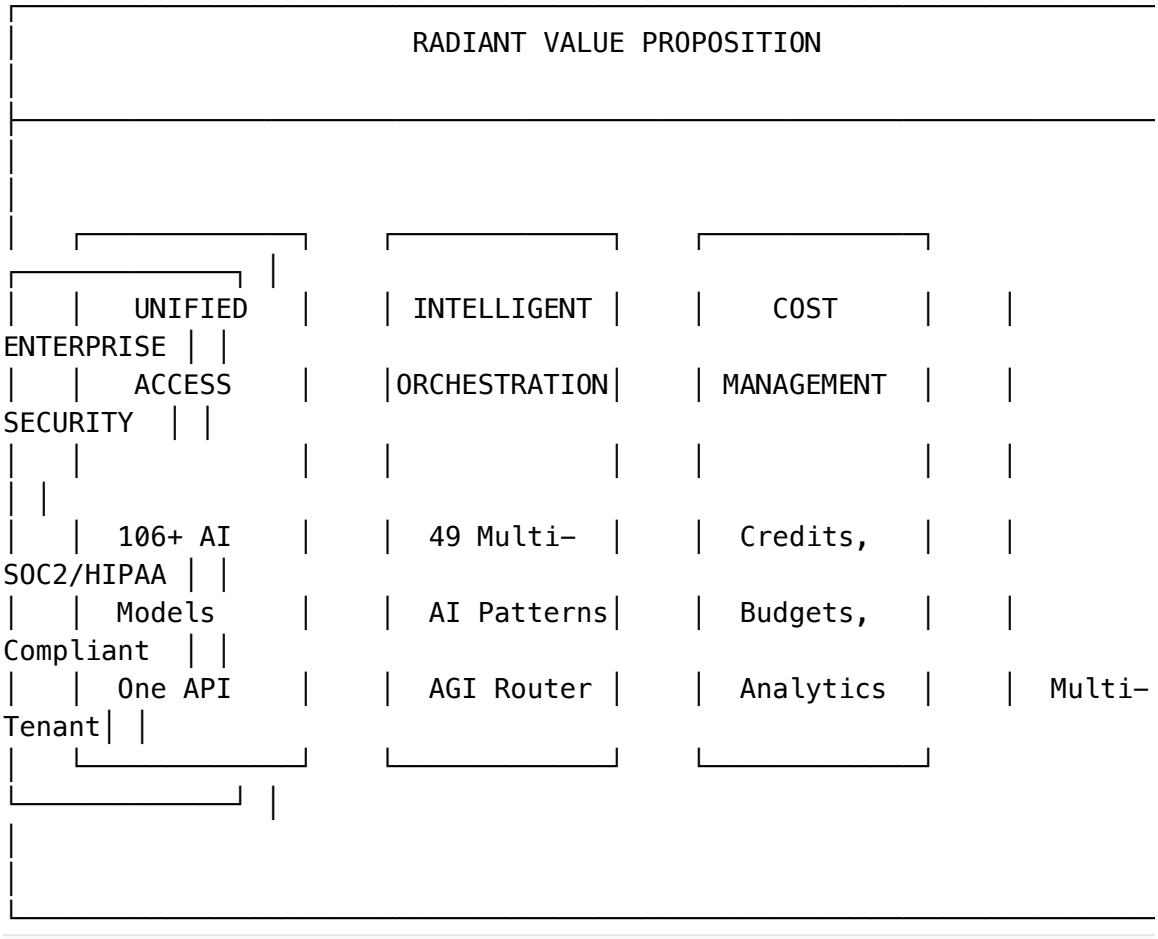
---

1. Platform Overview

1.1 What is RADIANT?

RADIANT (Real-time AI Distribution, Integration, and Automation Network for Tenants) is an enterprise-grade, multi-tenant SaaS platform that provides unified access to 106+ AI models across multiple providers, with intelligent orchestration, cost management, and comprehensive analytics.

1.2 Core Value Proposition



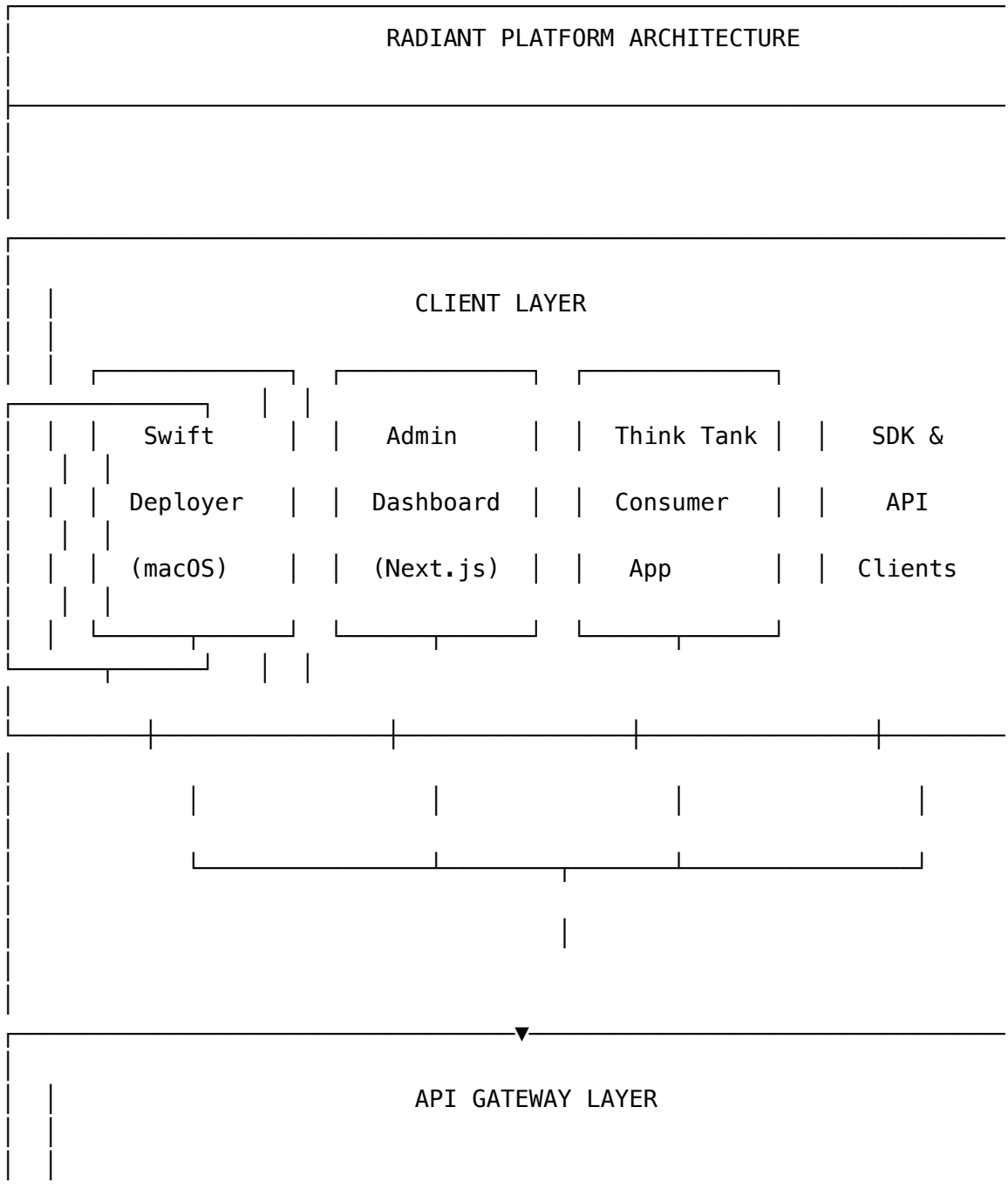
1.3 Platform Statistics

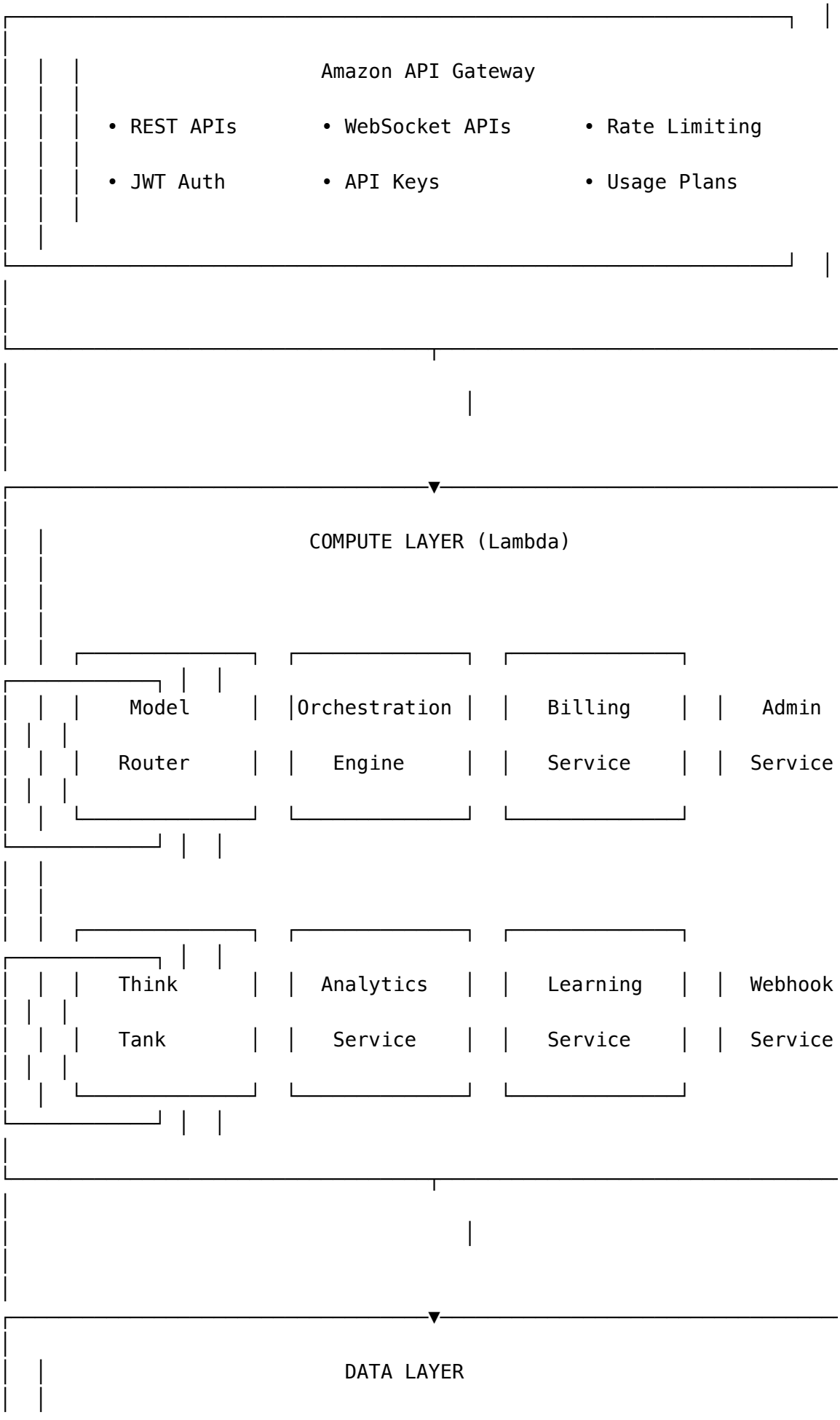
Metric	Value
AI Models Supported	106+ (50 external + 56 self-hosted)
AI Providers Integrated	15+ (OpenAI, Anthropic, Google, Meta, etc.)

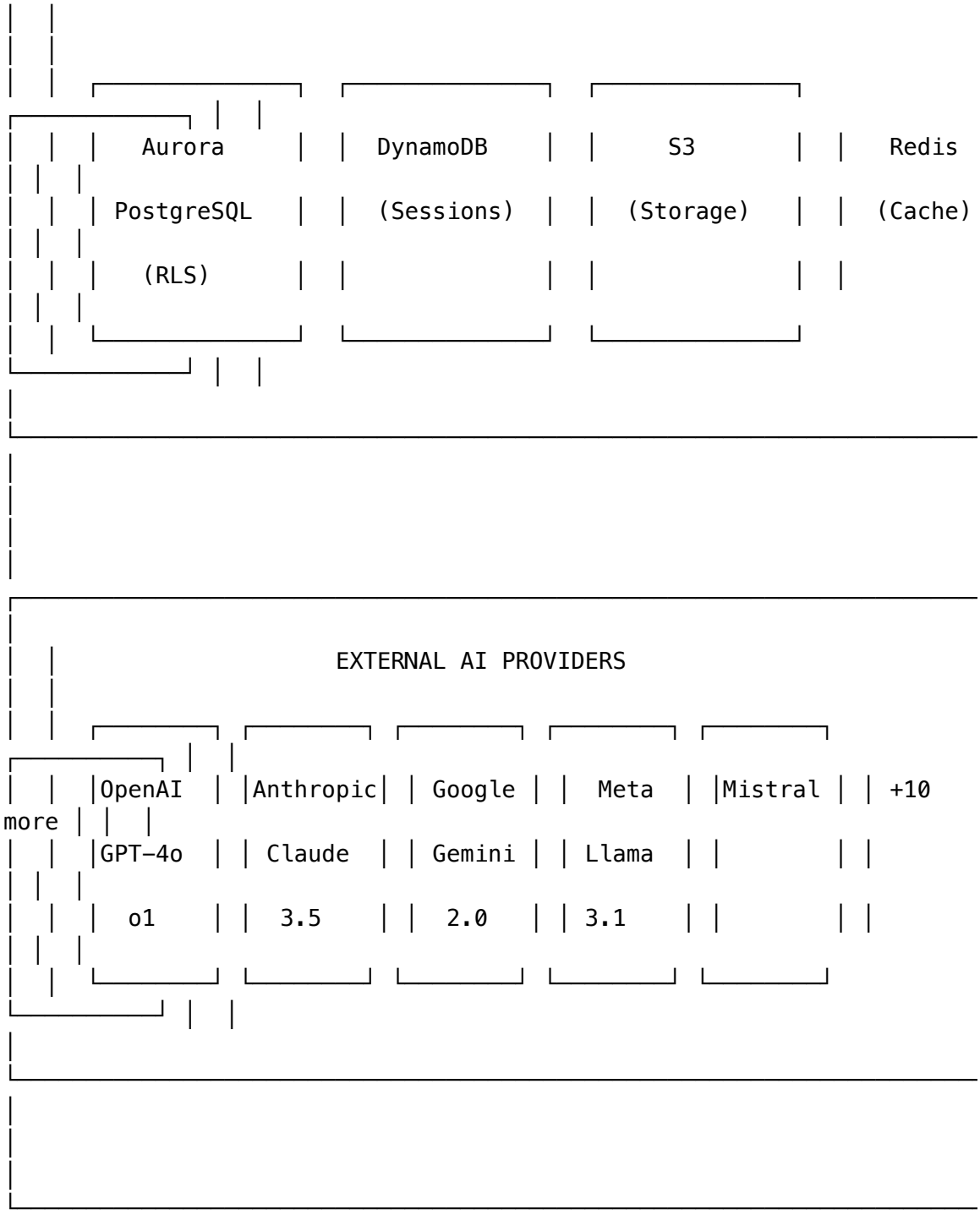
Metric	Value
Orchestration Patterns	49 documented patterns
Model Execution Modes	9 (thinking, research, fast, creative, etc.)
Database Migrations	66+ schema migrations
CDK Stacks	14 infrastructure stacks

2. System Architecture

2.1 High-Level Architecture

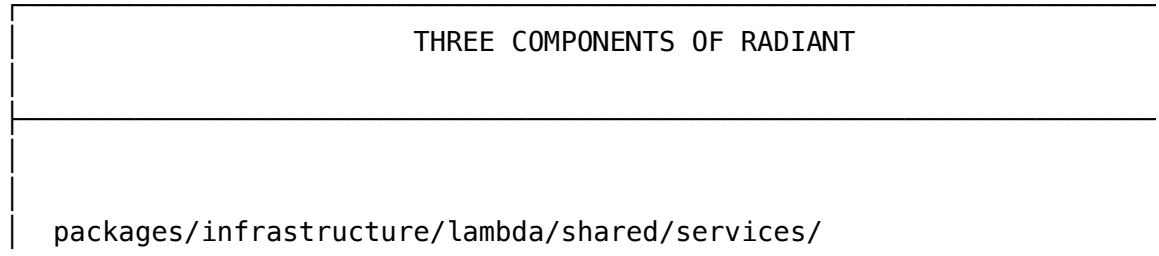


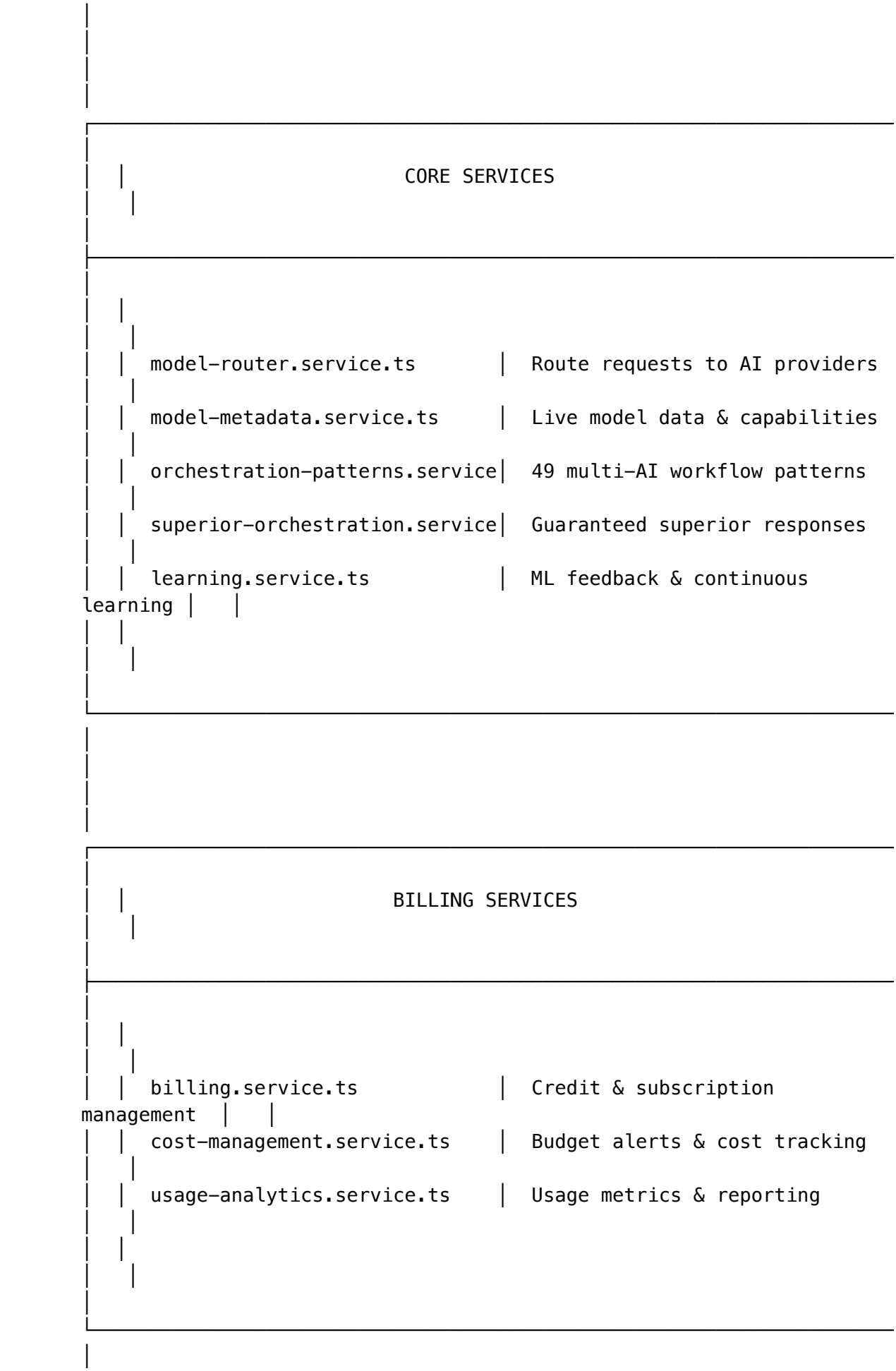

Amazon API Gateway
• REST APIs
• WebSocket APIs
• Rate Limiting
• JWT Auth
• API Keys
• Usage Plans
COMPUTE LAYER (Lambda)
Model Router
Orchestration Engine
Billing Service
Admin Service
Think Tank
Analytics Service
Learning Service
Webhook Service
DATA LAYER

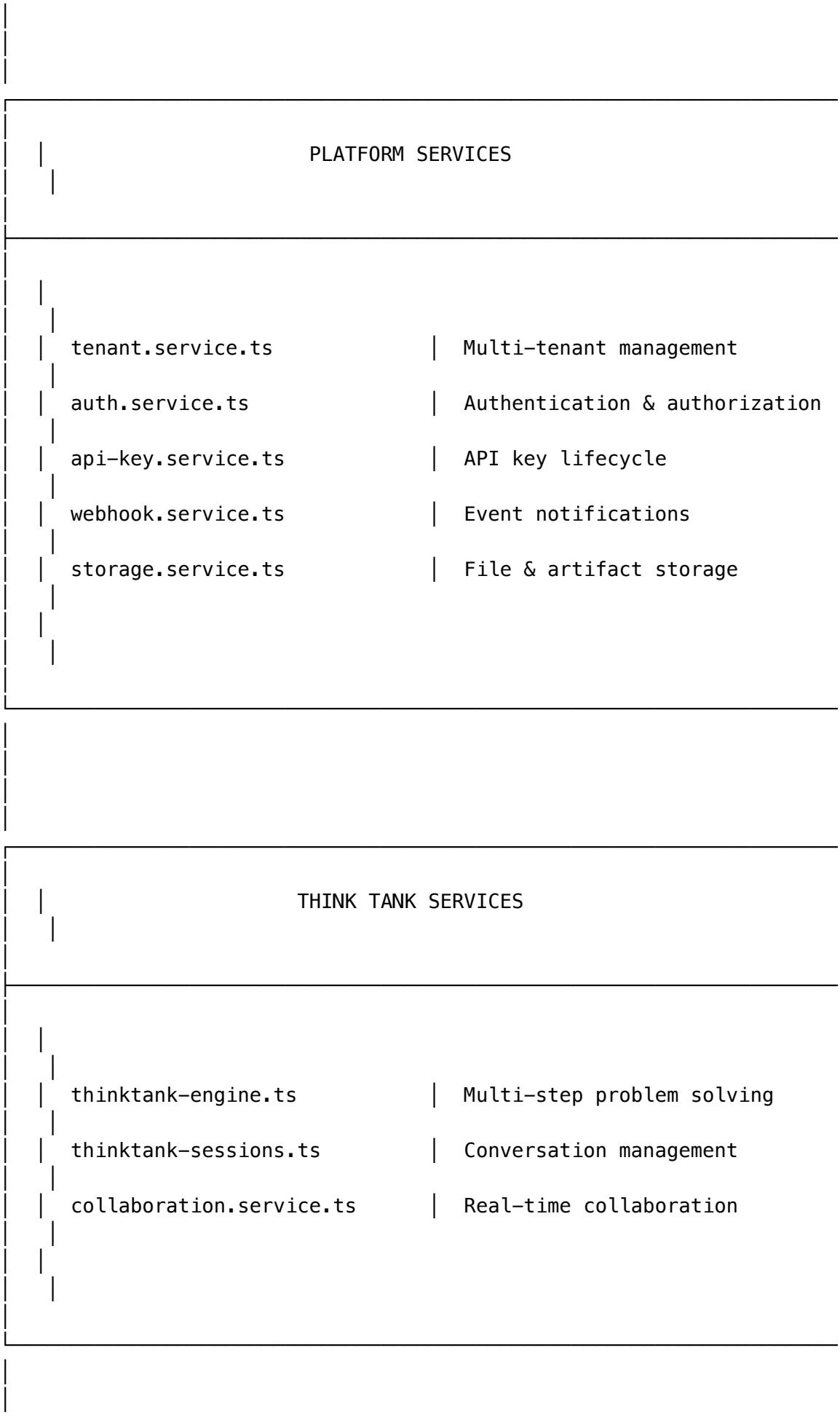


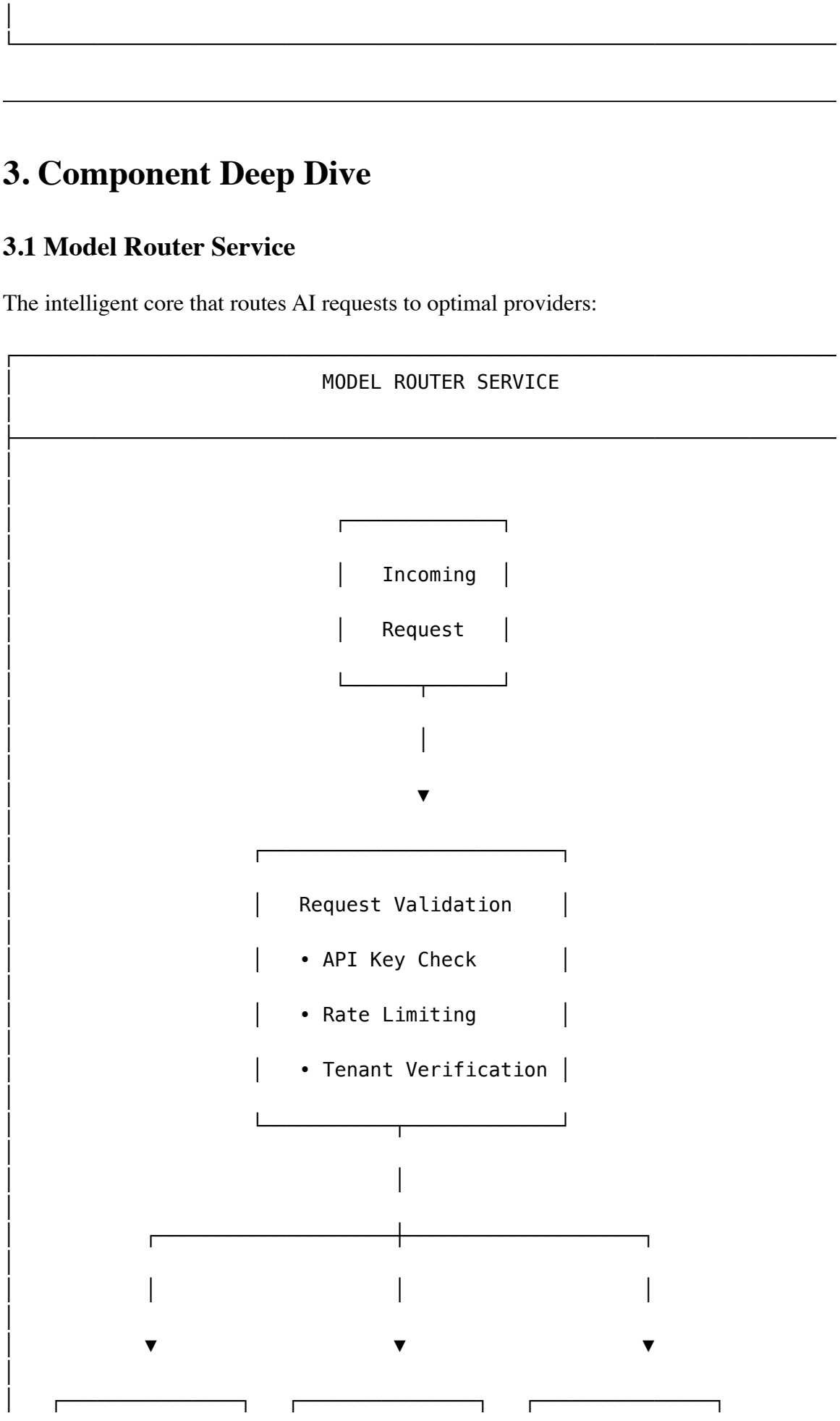
2.2 Three-Component Structure

RADIANT consists of three primary deployment components:




CORE SERVICES
model-router.service.ts | Route requests to AI providers
model-metadata.service.ts | Live model data & capabilities
orchestration-patterns.service | 49 multi-AI workflow patterns
superior-orchestration.service | Guaranteed superior responses
learning.service.ts | ML feedback & continuous learning
BILLING SERVICES
billing.service.ts | Credit & subscription management
cost-management.service.ts | Budget alerts & cost tracking
usage-analytics.service.ts | Usage metrics & reporting

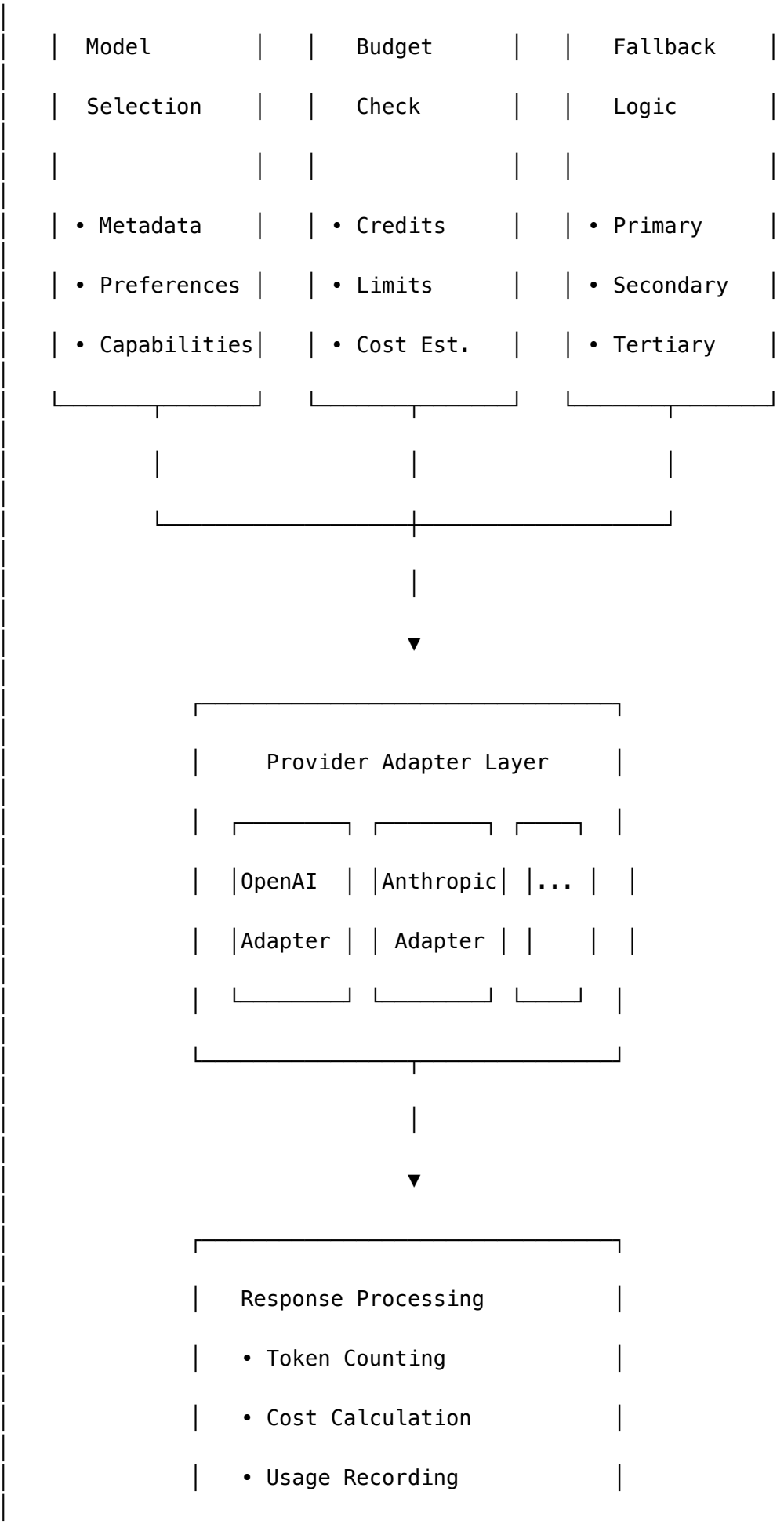

PLATFORM SERVICES
tenant.service.ts | Multi-tenant management
auth.service.ts | Authentication & authorization
api-key.service.ts | API key lifecycle
webhook.service.ts | Event notifications
storage.service.ts | File & artifact storage
THINK TANK SERVICES
thinktank-engine.ts | Multi-step problem solving
thinktank-sessions.ts | Conversation management
collaboration.service.ts | Real-time collaboration



3. Component Deep Dive

3.1 Model Router Service

The intelligent core that routes AI requests to optimal providers:


Model Selection
• Metadata
• Preferences
• Capabilities
Budget Check
• Credits
• Limits
• Cost Est.
Fallback Logic
• Primary
• Secondary
• Tertiary
Provider Adapter Layer
OpenAI Adapter
Anthropic Adapter
...
Response Processing
• Token Counting
• Cost Calculation
• Usage Recording

```
│               │   • Analytics Event             │
│
│               └─────────────────────────────────┘
│
│
│
└──────────────────────────────────────────────────────────────────────────────
```

3.2 Service Architecture

```
┌──────────────────────────────────────────────────────────────────────────────
│                         LAMBDA SERVICES ARCHITECTURE
│
├──────────────────────────────────────────────────────────────────────────────
│
│
│  packages/infrastructure/lambda/shared/services/
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────
│
│   │                         CORE SERVICES
│    │
│
├──────────────────────────────────────────────────────────────────────────────
│
│   │
│    │
│   │  model-router.service.ts       │  Route requests to AI providers
│    │
│   │  model-metadata.service.ts     │  Live model data & capabilities
│    │
│   │  orchestration-patterns.service│  49 multi-AI workflow patterns
│    │
│   │  superior-orchestration.service│  Guaranteed superior responses
│    │
│   │  learning.service.ts           │  ML feedback & continuous
learning │    │
│   │
│    │
│
└──────────────────────────────────────────────────────────────────────────────
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────
│
```

```
│   │                        BILLING SERVICES
│    │
│
├──────────────────────────────────────────────────────────────────────────────
│
│   │
│    │
│   │  billing.service.ts            │  Credit & subscription
management  │    │
│   │  cost-management.service.ts    │  Budget alerts & cost tracking
│    │
│   │  usage-analytics.service.ts    │  Usage metrics & reporting
│    │
│   │
│    │
│
└──────────────────────────────────────────────────────────────────────────────
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────
│
│   │                      PLATFORM SERVICES
│    │
│
├──────────────────────────────────────────────────────────────────────────────
│
│   │
│    │
│   │  tenant.service.ts             │  Multi-tenant management
│    │
│   │  auth.service.ts               │  Authentication & authorization
│    │
│   │  api-key.service.ts            │  API key lifecycle
│    │
│   │  webhook.service.ts            │  Event notifications
│    │
│   │  storage.service.ts            │  File & artifact storage
│    │
│   │
│    │
│
└──────────────────────────────────────────────────────────────────────────────
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────
│
│   │                      THINK TANK SERVICES
```

```
│    │
│
├──────────────────────────────────────────────────────────────────────────────┐
│
│   │
│    │
│   │  thinktank-engine.ts           │  Multi-step problem solving
│    │
│   │  thinktank-sessions.ts         │  Conversation management
│    │
│   │  collaboration.service.ts      │  Real-time collaboration
│    │
│   │
│    │
│
└──────────────────────────────────────────────────────────────────────────────┘
│
│
│
└──────────────────────────────────────────────────────────────────────────────┘
```

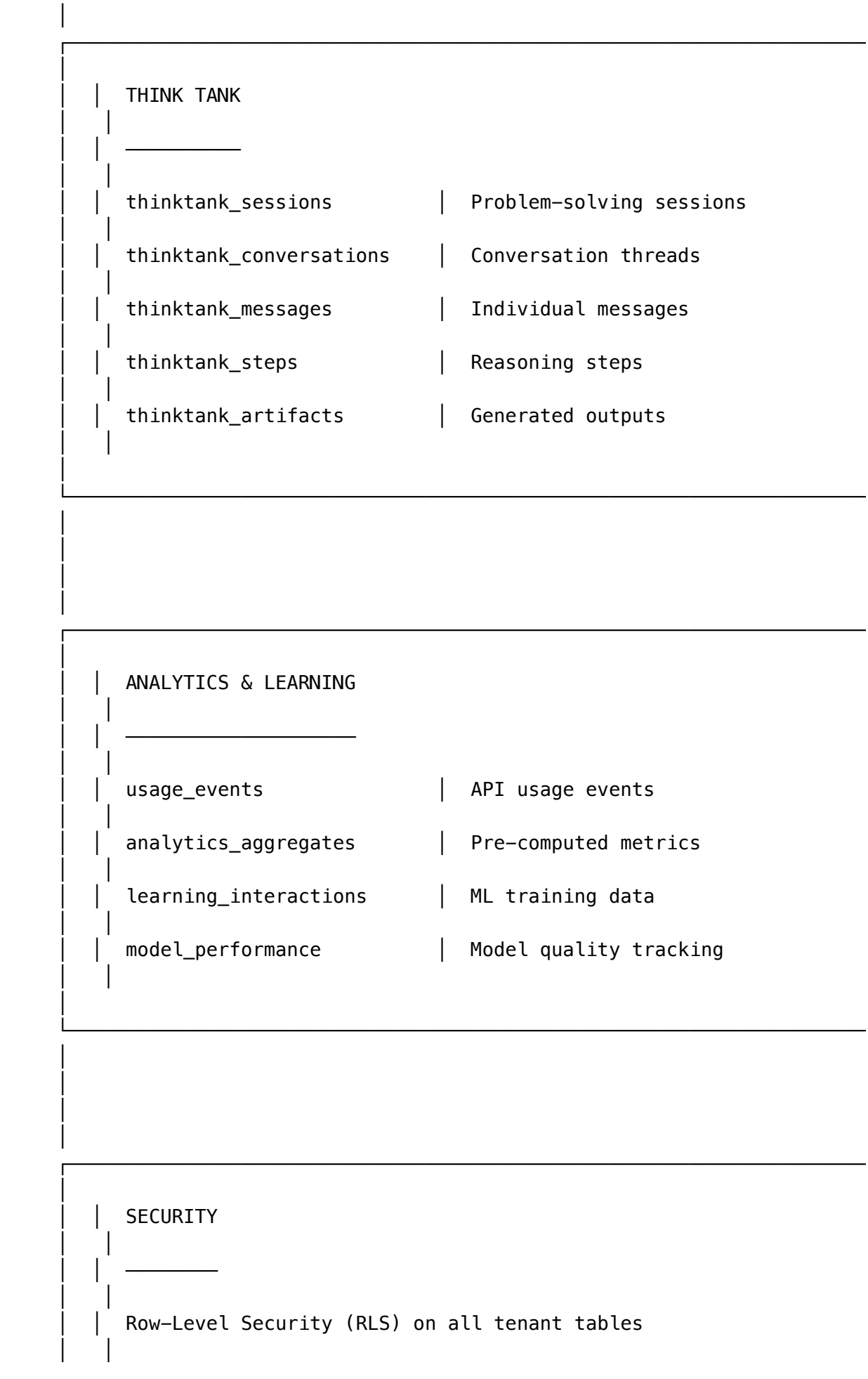
---

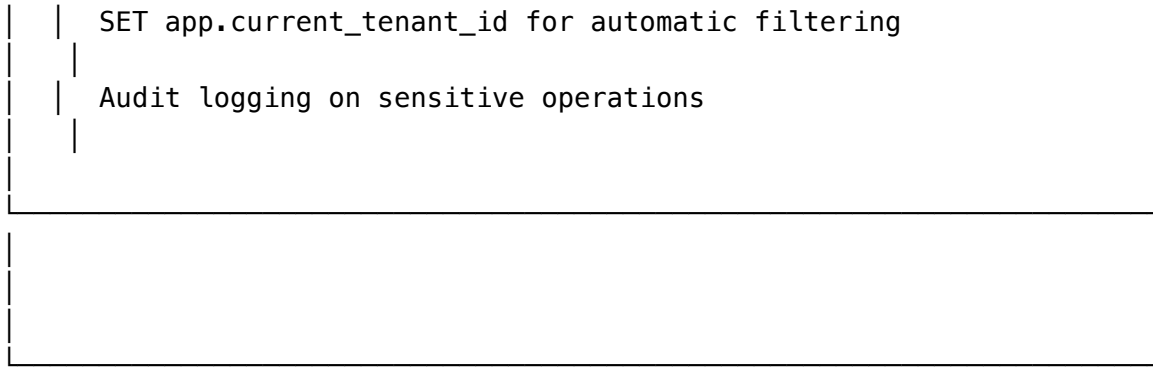
4. Data Architecture

4.1 Database Schema Overview

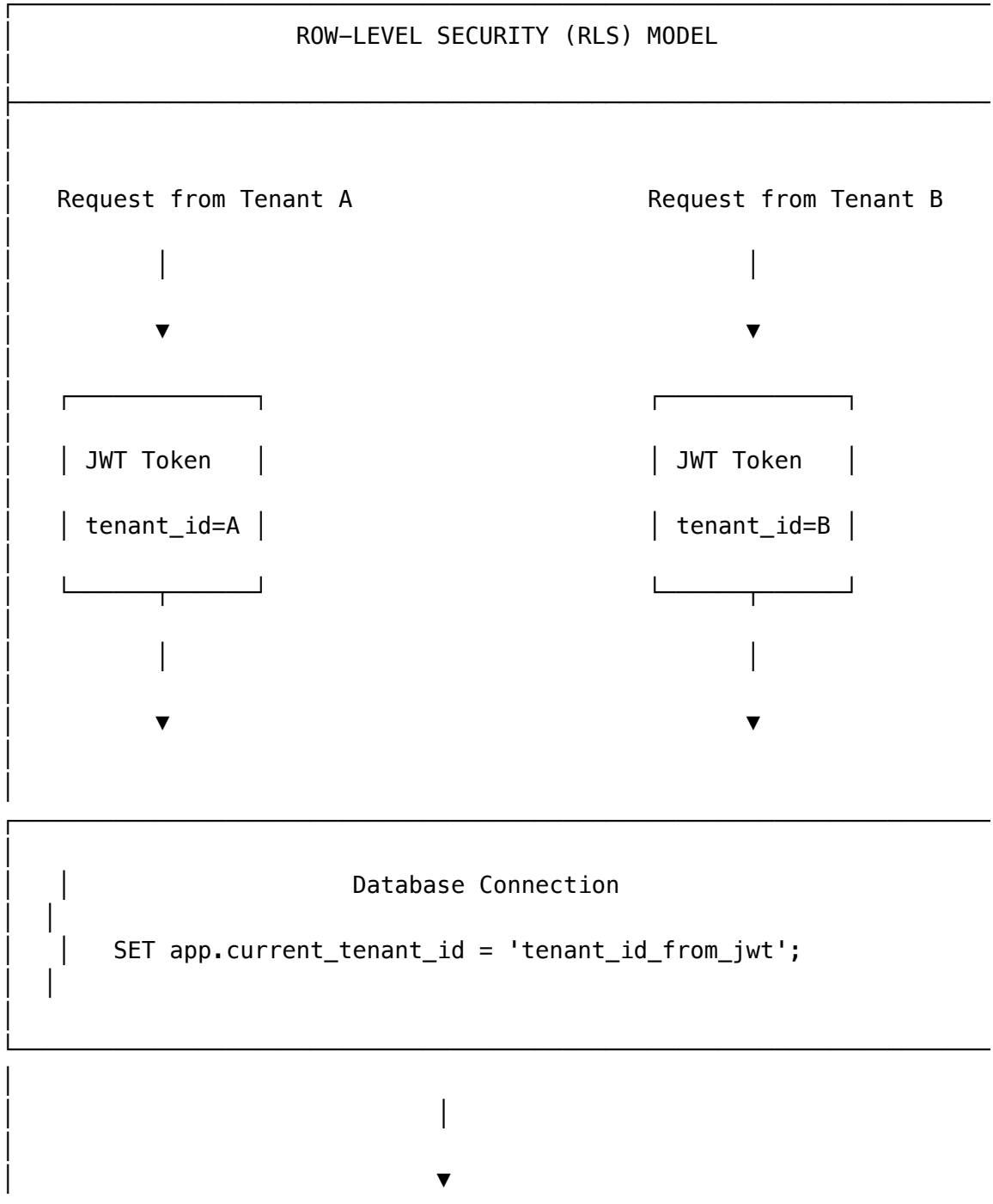
```
┌──────────────────────────────────────────────────────────────────────────────┐
│                    AURORA POSTGRESQL SCHEMA
│
├──────────────────────────────────────────────────────────────────────────────┐
│
│
│  66+ Migrations in packages/infrastructure/migrations/
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  CORE ENTITIES
│    │
│   │  ──────────────
│    │
│   │  tenants                    │  Multi-tenant organizations
│    │
│   │  users                      │  User accounts with roles
│    │
│   │  api_keys                   │  API authentication keys
│    │
```

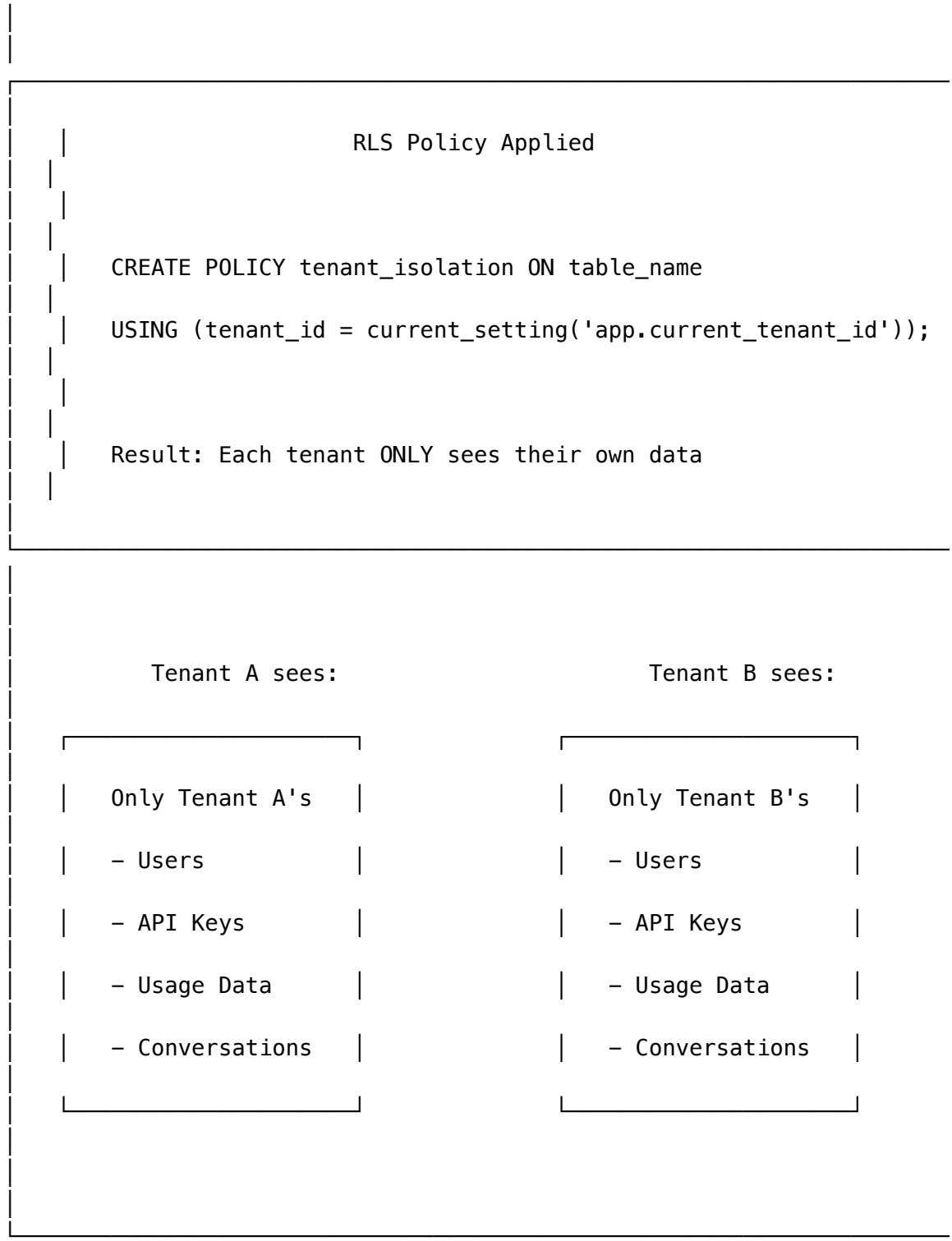
```
│   │  model_configurations       │  Per-tenant model settings
│   │
│   │  model_metadata             │  AI model capabilities & pricing
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
                                    │
                                    │
                                    │
                                    │
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  BILLING & CREDITS
│   │
│   │  ──────────────────
│   │
│   │  credit_accounts            │  Tenant credit balances
│   │
│   │  credit_transactions        │  Credit usage history
│   │
│   │  subscriptions              │  Plan subscriptions
│   │
│   │  invoices                   │  Billing invoices
│   │
│   │  budgets                    │  Spending limits & alerts
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
                                    │
                                    │
                                    │
                                    │
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  ORCHESTRATION
│   │
│   │  ──────────────
│   │
│   │  orchestration_methods      │  Reusable AI method definitions
│   │
│   │  orchestration_workflows    │  49 workflow patterns
│   │
│   │  workflow_method_bindings   │  Steps linking workflows to methods
│   │
│   │  orchestration_executions   │  Execution history & results
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
                                    │
                                    │
                                    │
```


THINK TANK
thinktank_sessions | Problem-solving sessions
thinktank_conversations | Conversation threads
thinktank_messages | Individual messages
thinktank_steps | Reasoning steps
thinktank_artifacts | Generated outputs
ANALYTICS & LEARNING
usage_events | API usage events
analytics_aggregates | Pre-computed metrics
learning_interactions | ML training data
model_performance | Model quality tracking
SECURITY
Row-Level Security (RLS) on all tenant tables



4.2 Multi-Tenant Data Isolation

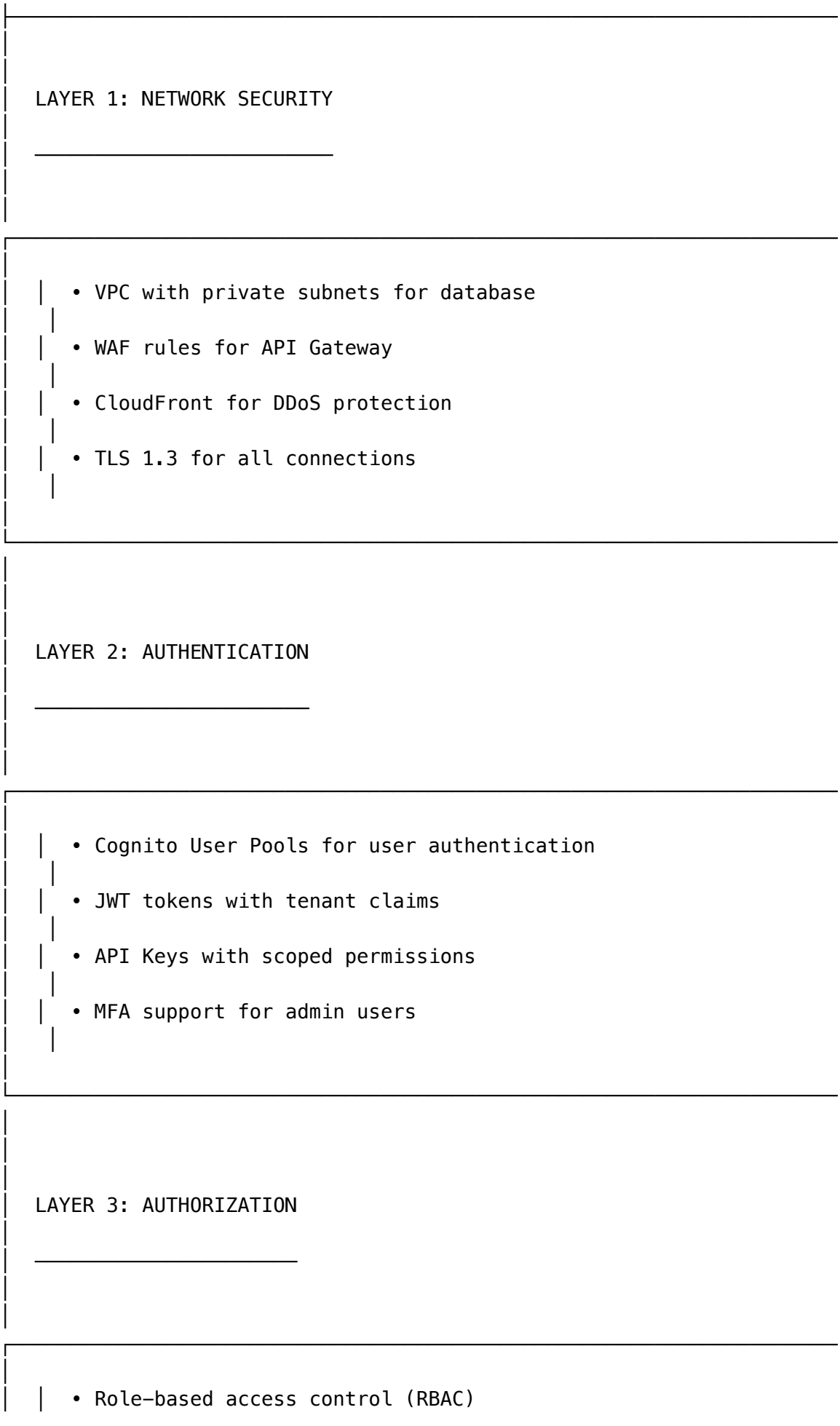


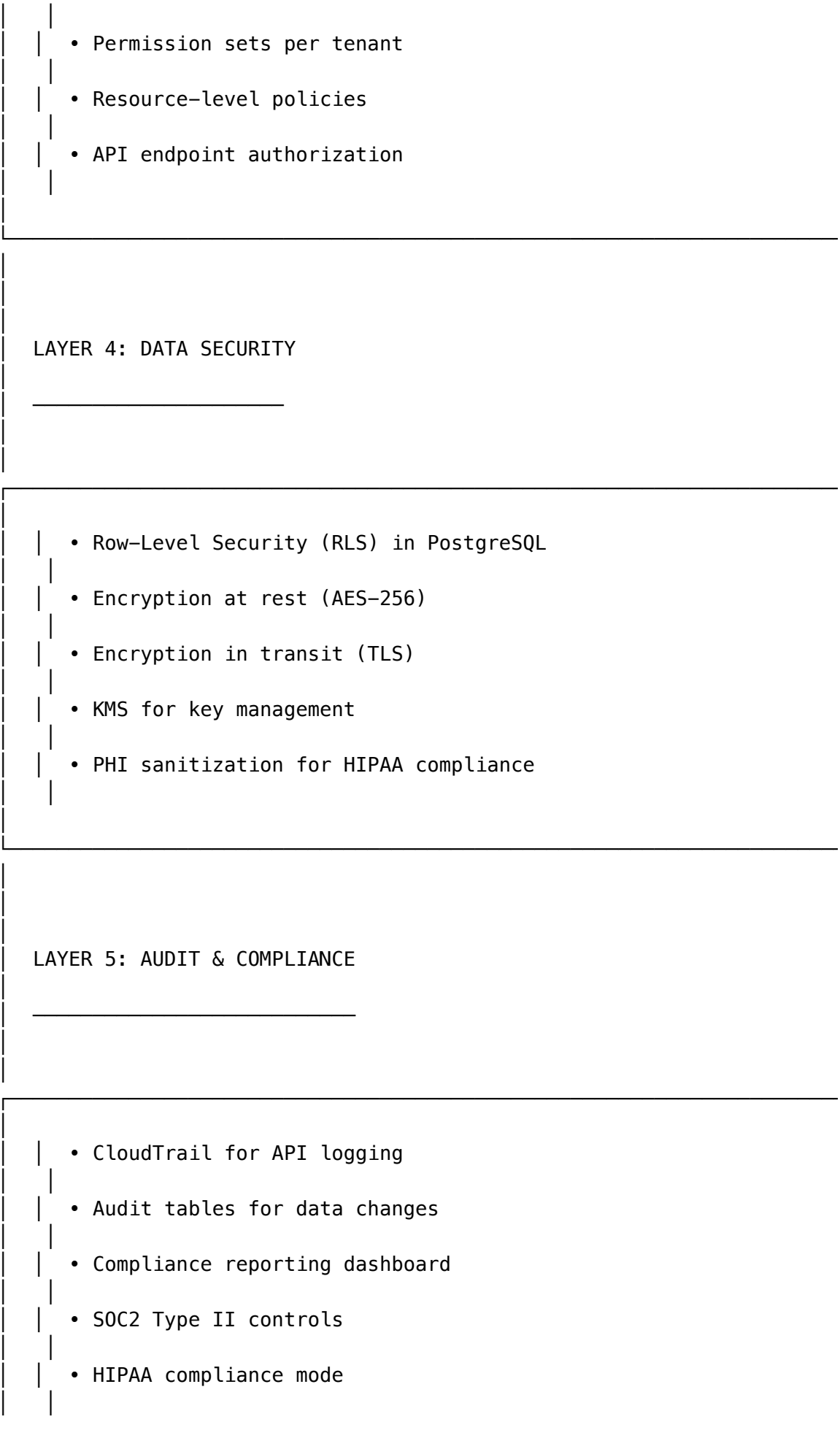


---

5. Security Architecture

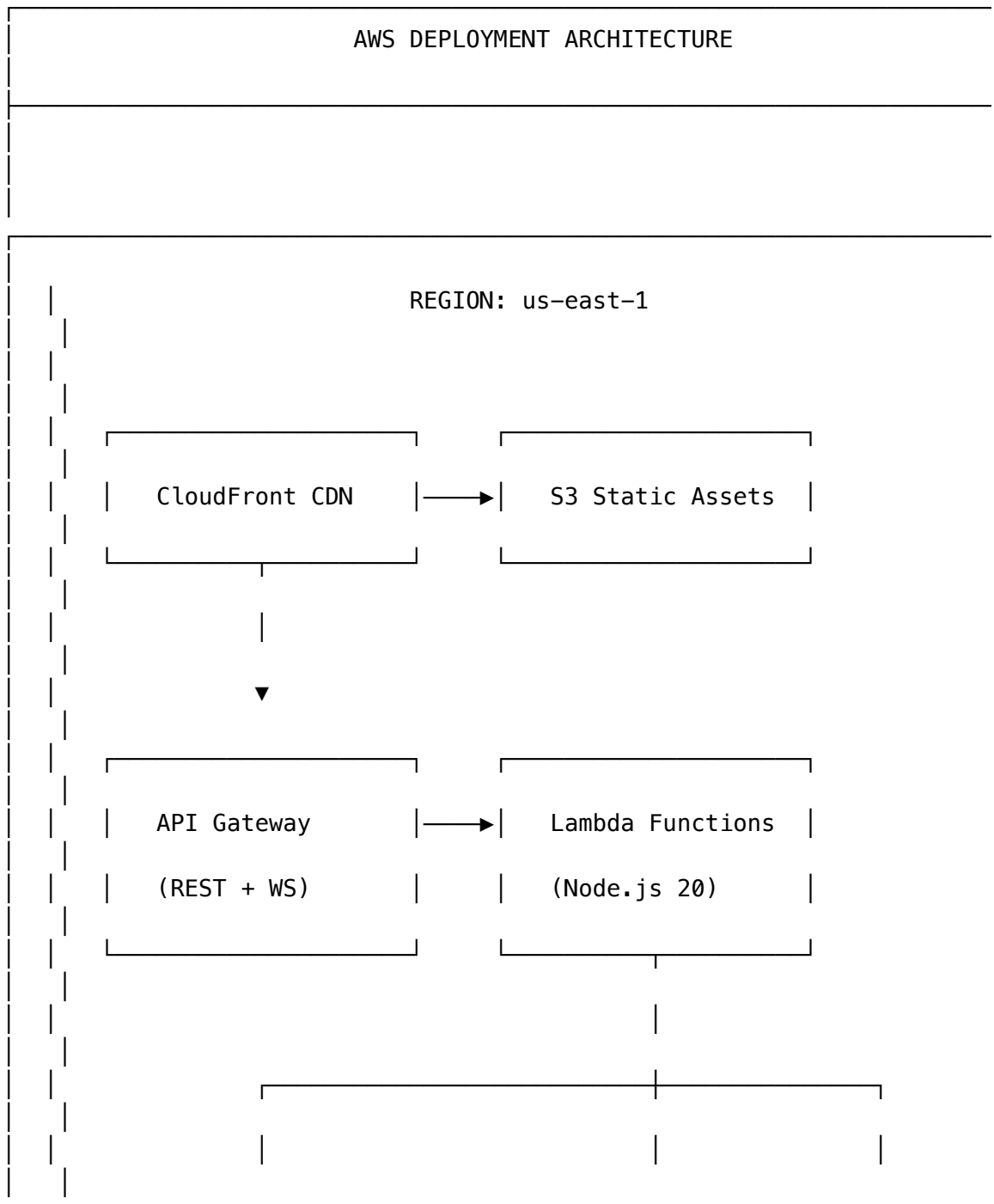
5.1 Security Layers

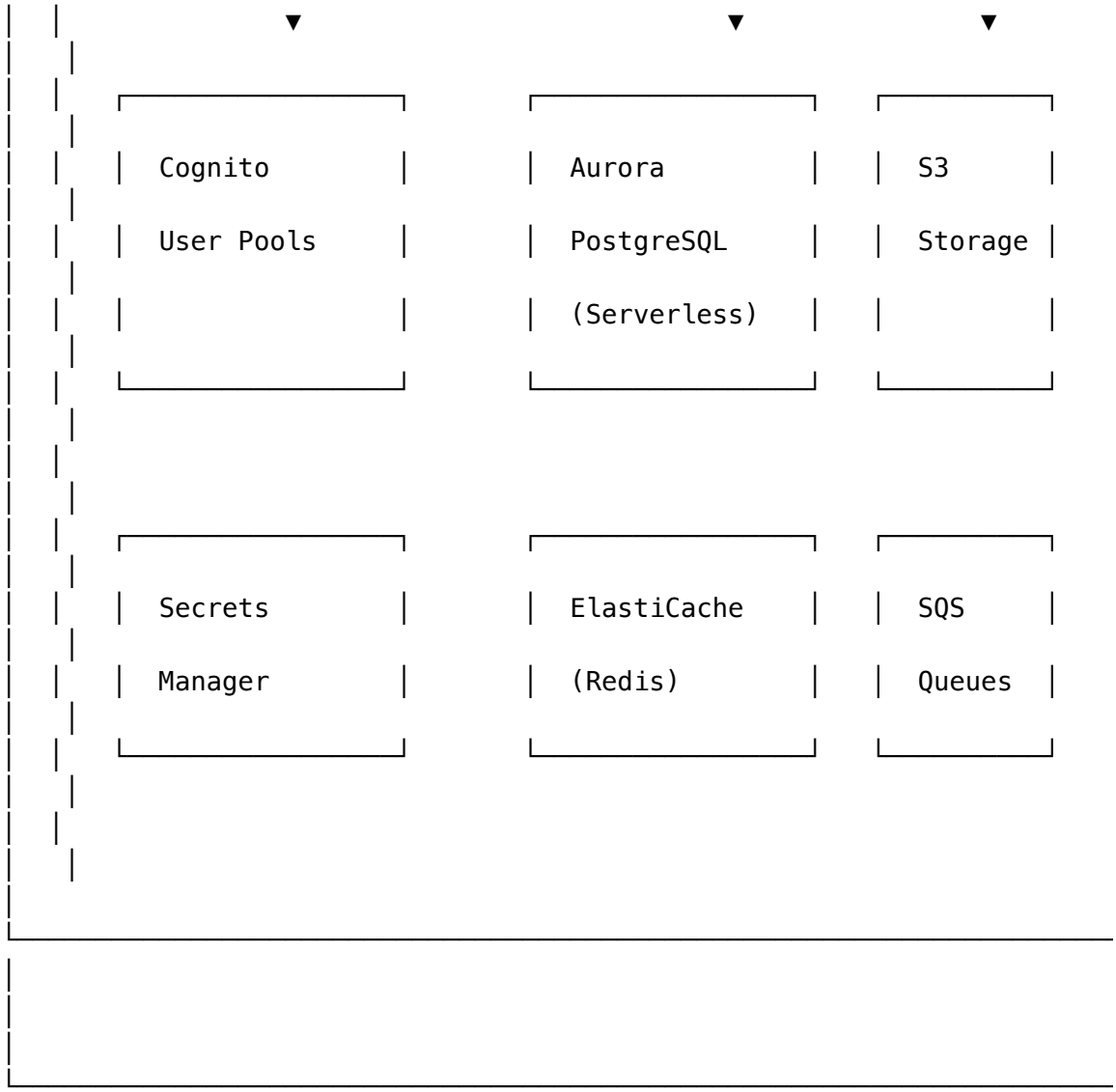

LAYER 1: NETWORK SECURITY
• VPC with private subnets for database
• WAF rules for API Gateway
• CloudFront for DDoS protection
• TLS 1.3 for all connections
LAYER 2: AUTHENTICATION
• Cognito User Pools for user authentication
• JWT tokens with tenant claims
• API Keys with scoped permissions
• MFA support for admin users
LAYER 3: AUTHORIZATION
• Role-based access control (RBAC)


• Permission sets per tenant
• Resource-level policies
• API endpoint authorization
LAYER 4: DATA SECURITY
• Row-Level Security (RLS) in PostgreSQL
• Encryption at rest (AES-256)
• Encryption in transit (TLS)
• KMS for key management
• PHI sanitization for HIPAA compliance
LAYER 5: AUDIT & COMPLIANCE
• CloudTrail for API logging
• Audit tables for data changes
• Compliance reporting dashboard
• SOC2 Type II controls
• HIPAA compliance mode

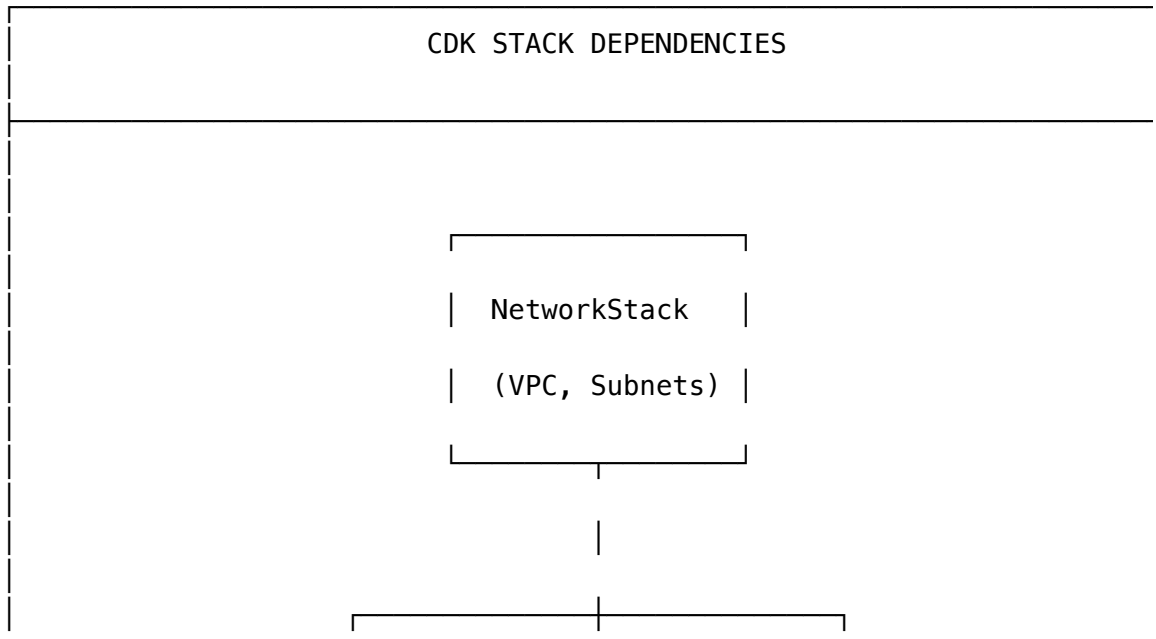
6. Deployment Architecture

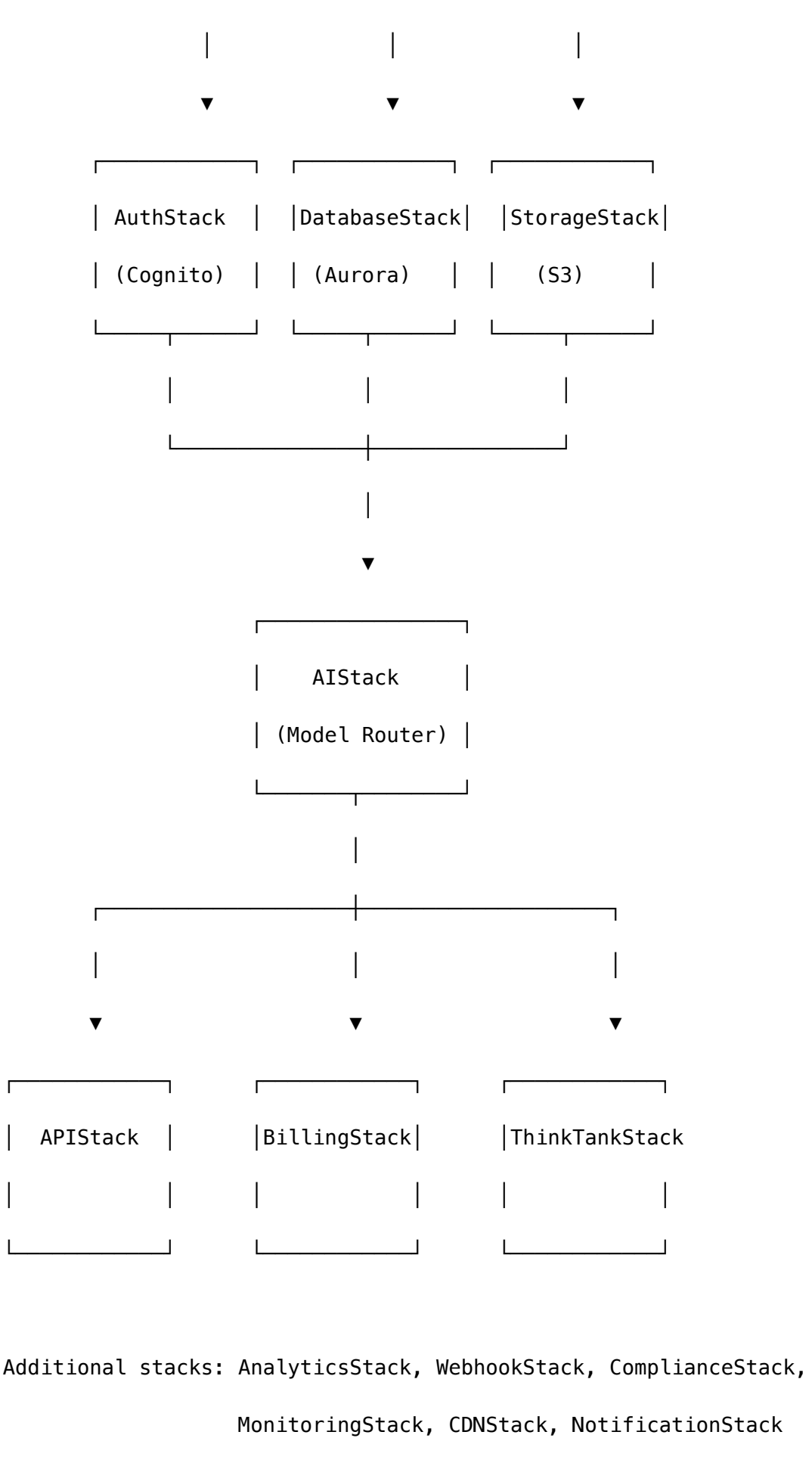
6.1 AWS Infrastructure

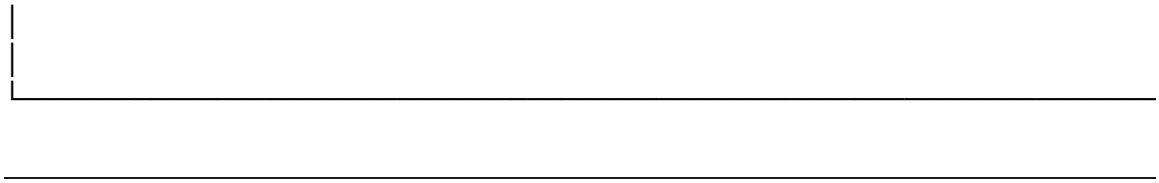




6.2 CDK Stack Dependencies



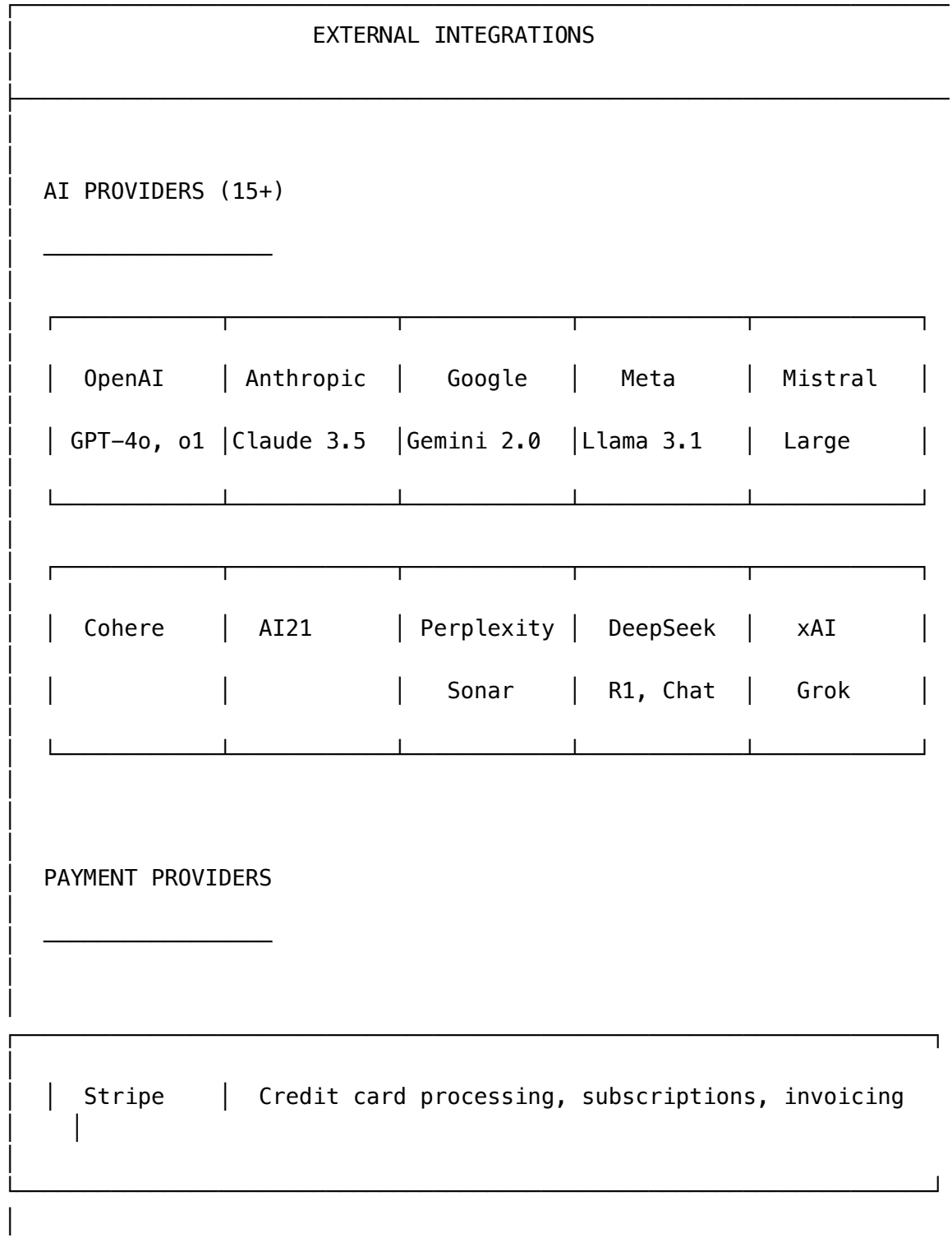

AuthStack
(Cognito)
DatabaseStack
(Aurora)
StorageStack
(S3)
AIStack
(Model Router)
APIStack
BillingStack
ThinkTankStack
Additional stacks: AnalyticsStack, WebhookStack, ComplianceStack,
MonitoringStack, CDNStack, NotificationStack

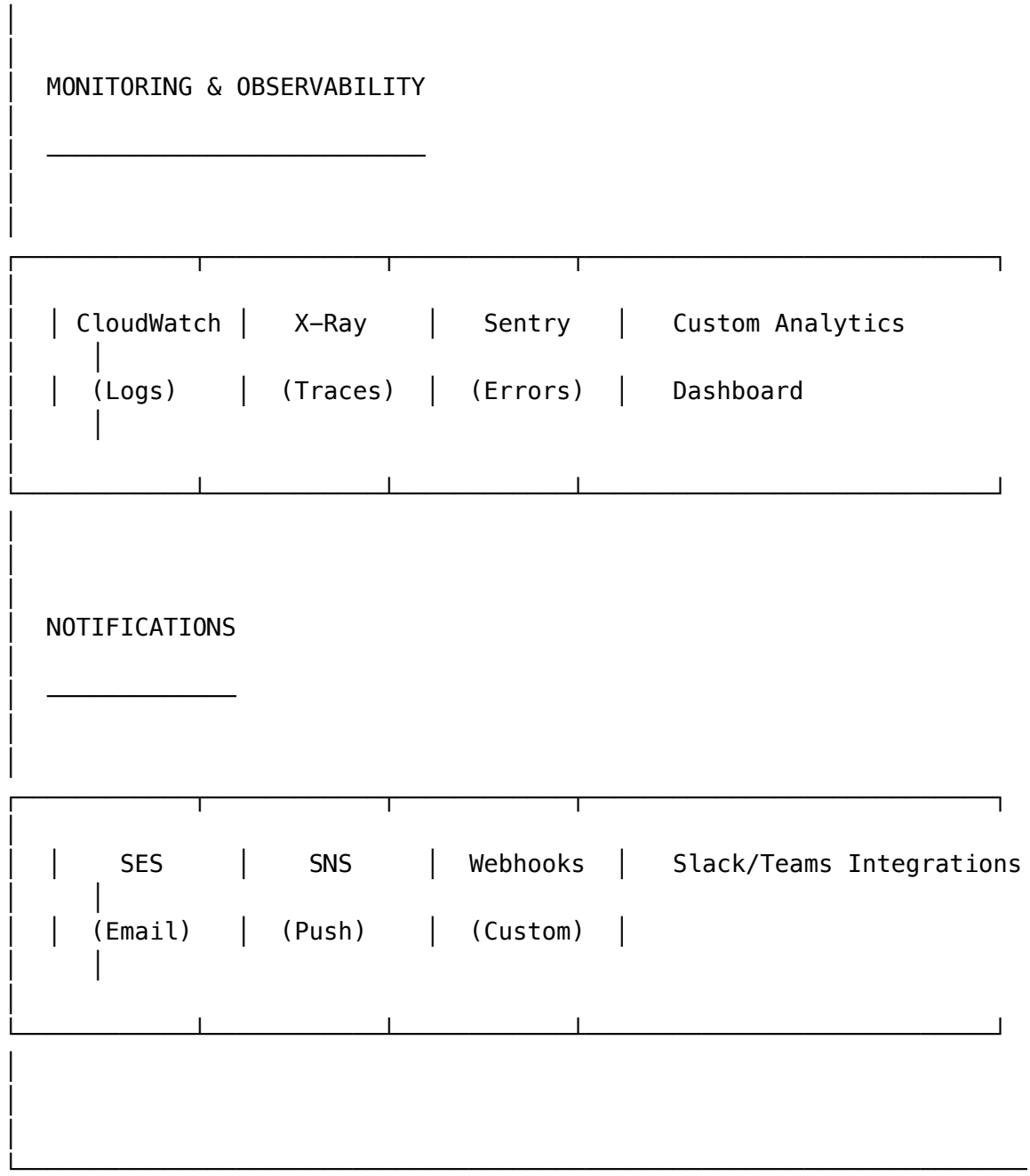


---

7. Integration Points

7.1 External API Integrations





RADIANT Platform Architecture v4.18.0

Building the future of enterprise AI

© 2024 RADIANT. All Rights Reserved.

Think Tank Platform Architecture

Advanced Multi-Step AI Problem Solving

Version 3.2.0 | December 2024

A comprehensive technical architecture document for the Think Tank AI reasoning platform

---

Table of Contents

1. Platform Overview
2. Core Architecture
3. Problem Solving Pipeline
4. Session Management
5. Collaboration Features
6. Domain Modes
7. Quality & Confidence
8. User Interface

---

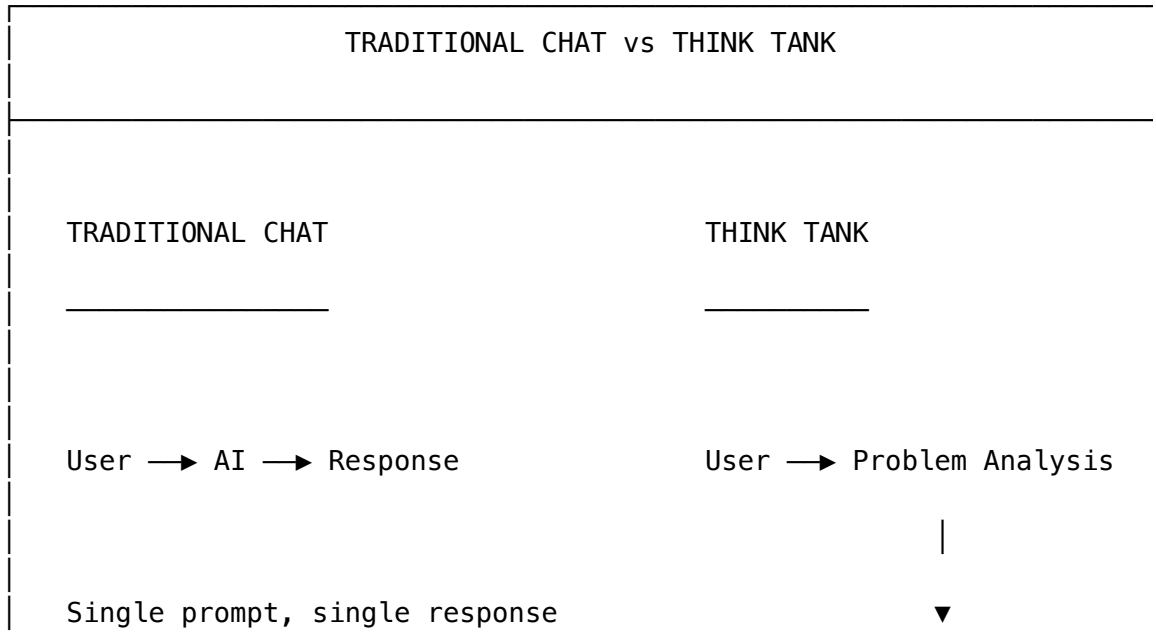
1. Platform Overview

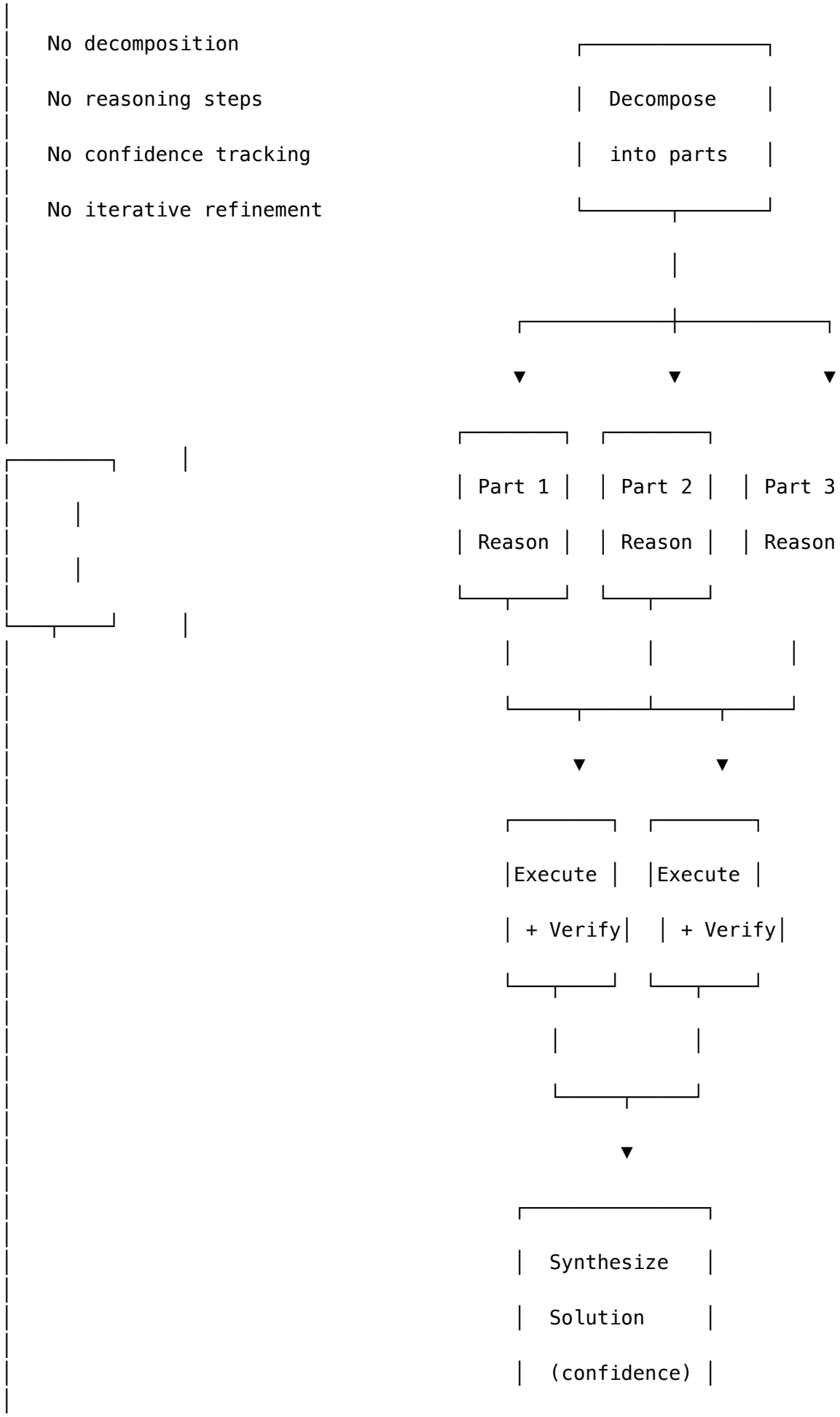
1.1 What is Think Tank?

Think Tank is an advanced AI reasoning platform that decomposes complex problems into manageable sub-problems, applies multi-step reasoning, and synthesizes comprehensive solutions using orchestrated AI models.

Unlike simple chat interfaces, Think Tank: - **Decomposes** complex problems into sub-tasks - **Reasons** through each component step-by-step - **Executes** specialized AI calls for each step - **Synthesizes** results into coherent solutions - **Tracks** confidence and quality throughout

1.2 Think Tank vs Traditional Chat



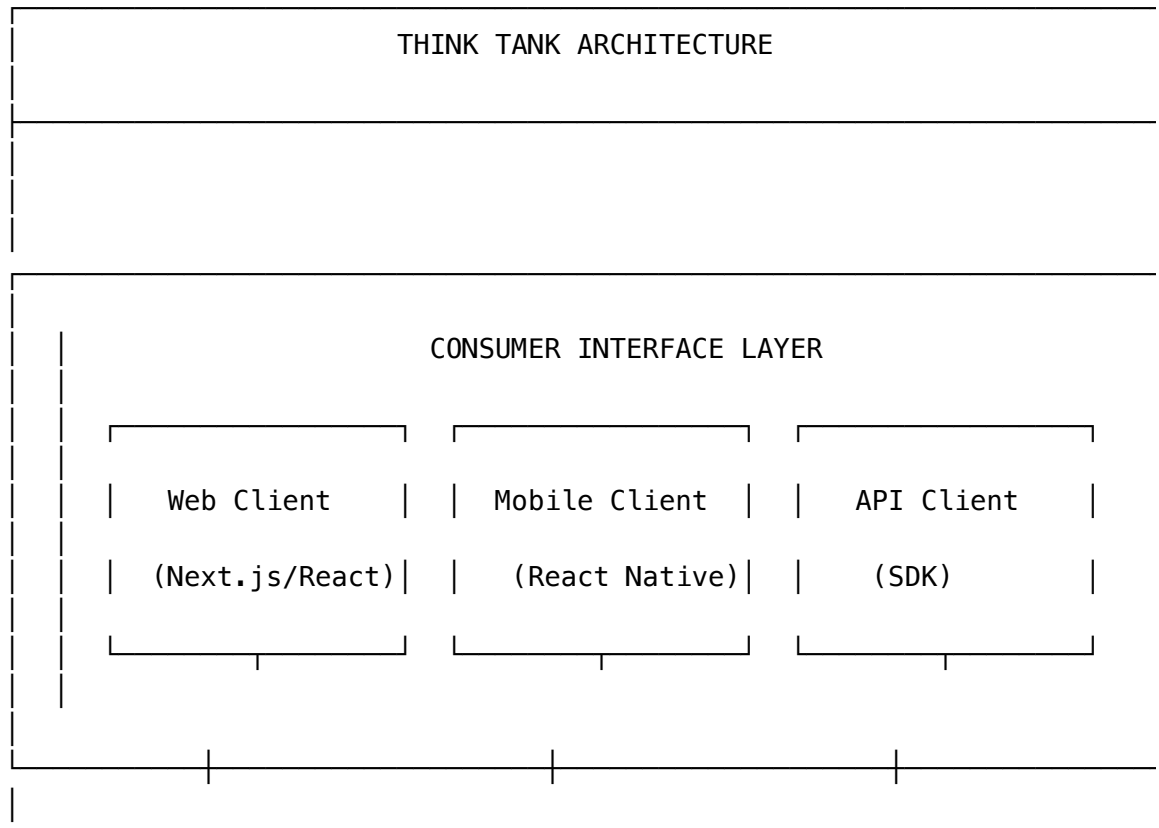

No decomposition
No reasoning steps
No confidence tracking
No iterative refinement
Decompose into parts
Part 1 Reason
Part 2 Reason
Part 3 Reason
Execute + Verify
Execute + Verify
Synthesize Solution (confidence)

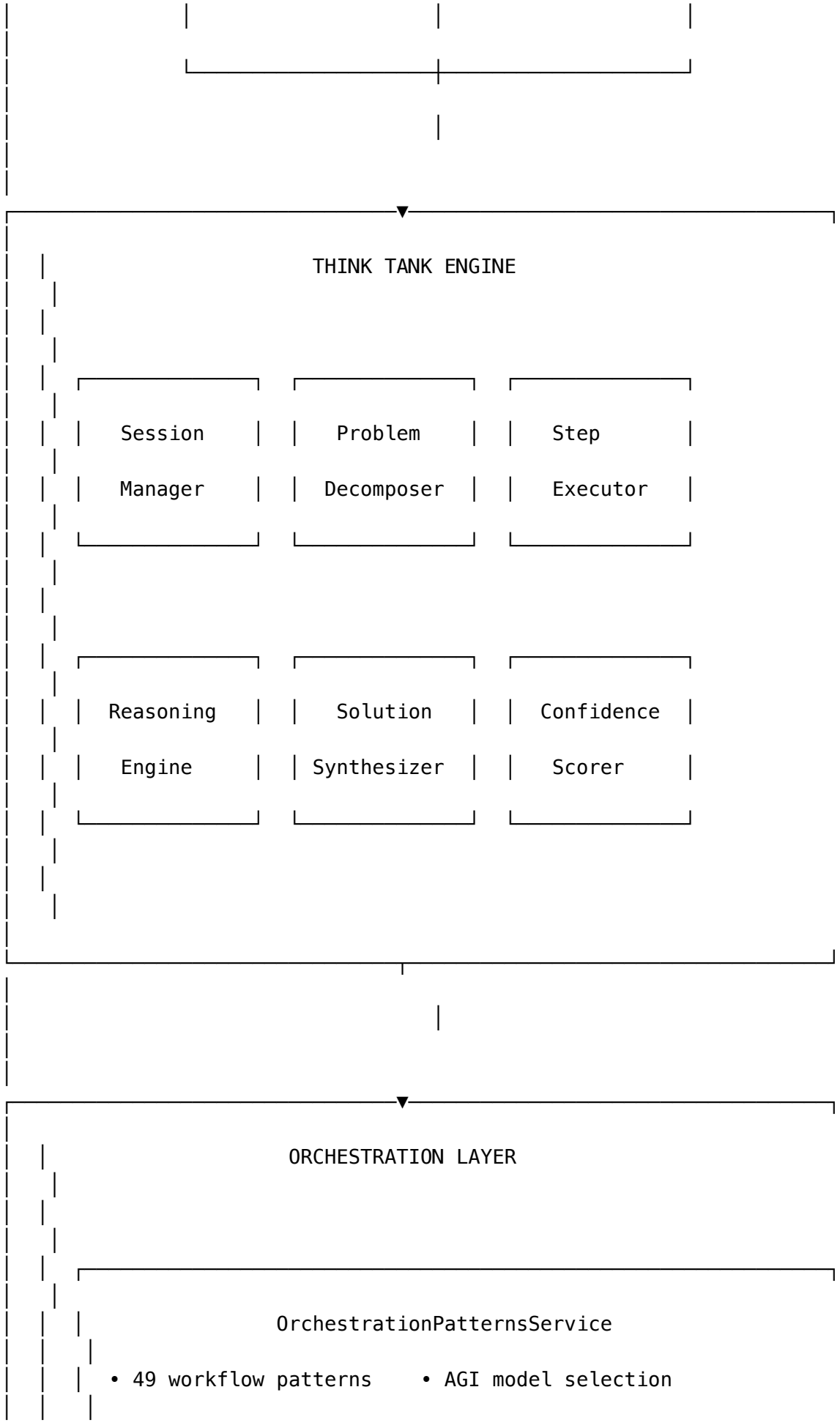
1.3 Key Capabilities

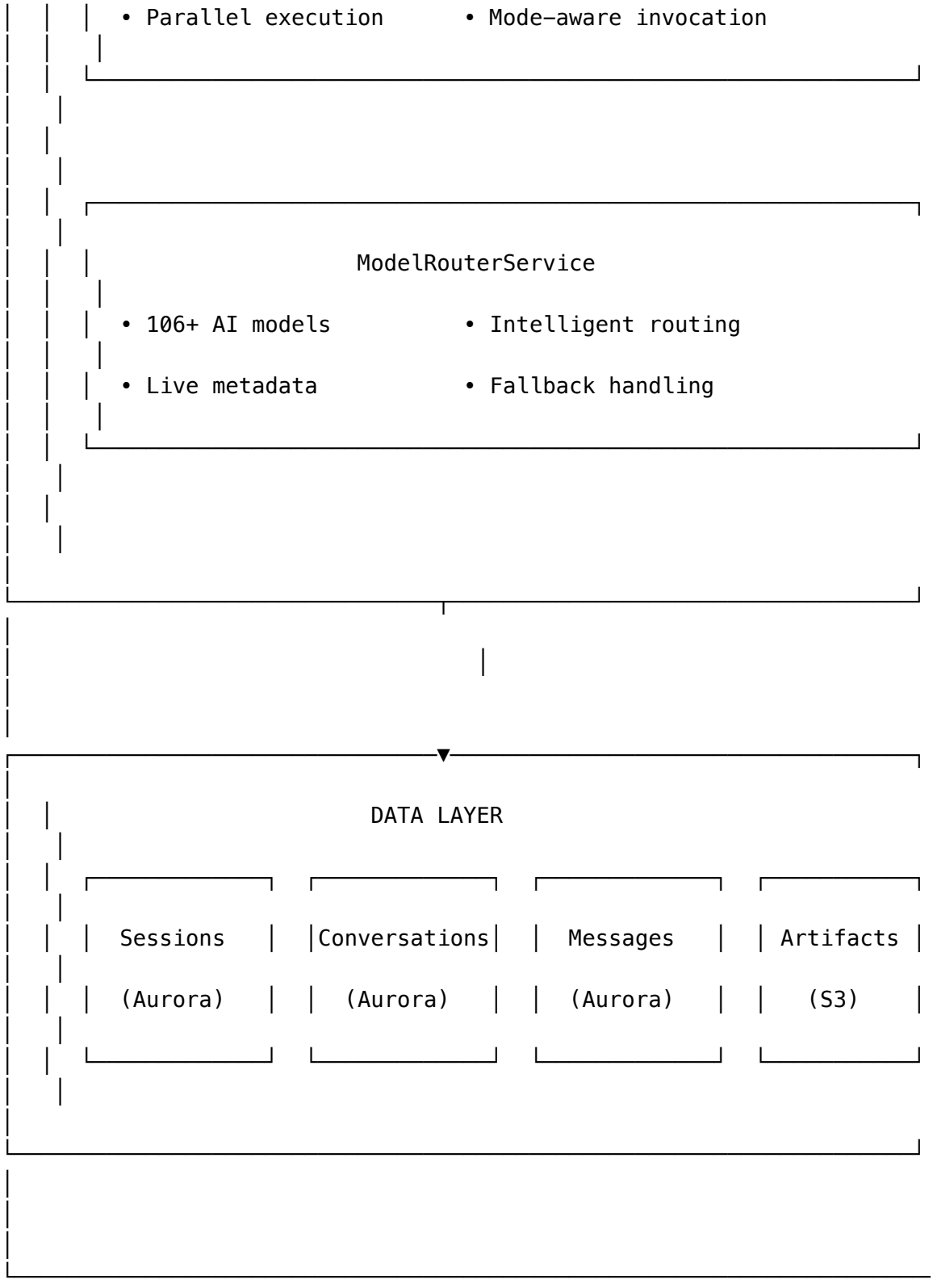
Capability	Description
Problem Decomposition	Breaks complex questions into manageable sub-problems
Multi-Step Reasoning	Chain-of-thought with recorded steps
Domain Specialization	8+ specialized reasoning modes
Confidence Tracking	Quality scores for every step
Artifact Generation	Code, documents, diagrams as outputs
Real-time Collaboration	Multiple users solving together
Session Persistence	Resume any session later
Cost Transparency	Token and cost tracking per step

2. Core Architecture

2.1 System Components

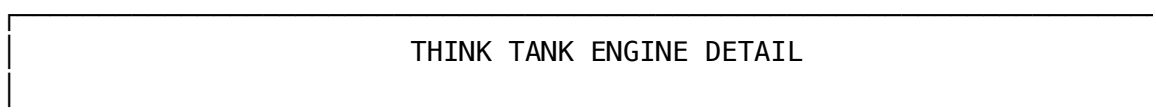



THINK TANK ENGINE
Session Manager
Problem Decomposer
Step Executor
Reasoning Engine
Solution Synthesizer
Confidence Scorer
ORCHESTRATION LAYER
OrchestrationPatternsService
• 49 workflow patterns
• AGI model selection

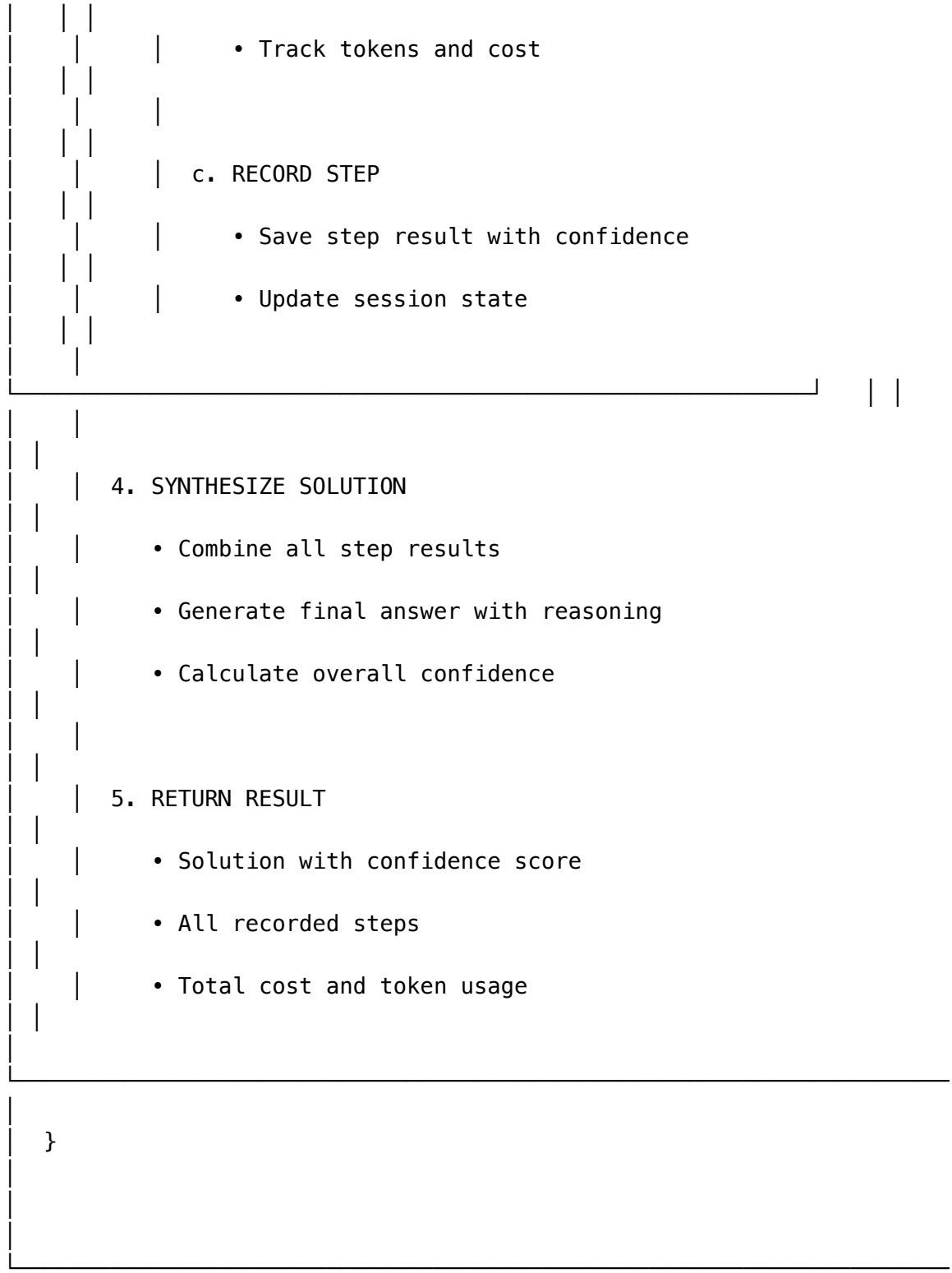


2.2 Think Tank Engine

The core engine that powers intelligent problem solving:



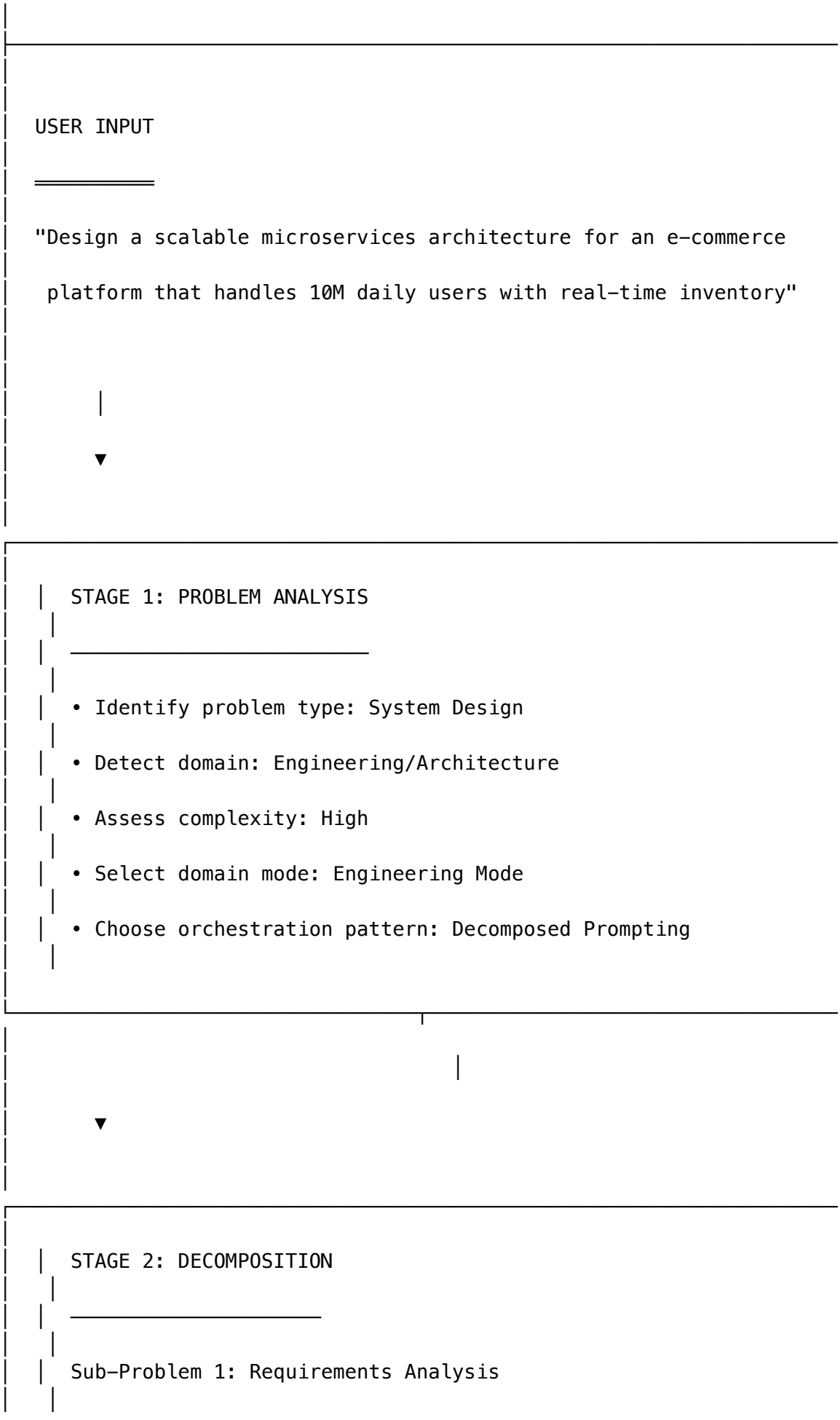
```
├──────────────────────────────────────────────────────────────────────────────────────────────┤
│                                                                                              │
│                                                                                              │
│  class ThinkTankEngine {                                                                     │
│                                                                                              │
│                                                                                              │
│                                                                                              │
│                                                                                              │
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│                                                                                              │
│    │  async solve(problem: ThinkTankProblem):
Promise<ThinkTankResult>   │ │
│    │                                                                                         │
│ │
│    │  1. CREATE SESSION                                                                      │
│ │
│    │     • Initialize session with problem context                                           │
│ │
│    │     • Set domain mode and preferences                                                   │
│ │
│    │     • Record start time and user info                                                   │
│ │
│    │                                                                                         │
│ │
│    │  2. DECOMPOSE PROBLEM                                                                   │
│ │
│    │     • AI analyzes problem structure                                                     │
│ │
│    │     • Identifies sub-problems and dependencies                                          │
│ │
│    │     • Creates execution plan                                                            │
│ │
│    │                                                                                         │
│ │
│    │  3. FOR EACH SUB-PROBLEM:                                                               │
│ │
│    │                                                                                         │
┌──────────────────────────────────────────────────────────────────────────────────────┐   │ │
│    │     │  a. REASON                                                                  │
│   │ │
│    │     │     • Chain-of-thought analysis                                             │
│   │ │
│    │     │     • Record reasoning steps                                                │
│   │ │
│    │     │                                                                             │
│   │ │
│    │     │  b. EXECUTE                                                                 │
│   │ │
│    │     │     • Call appropriate AI model(s)                                          │
│   │ │
│    │     │     • May use parallel execution                                            │
```

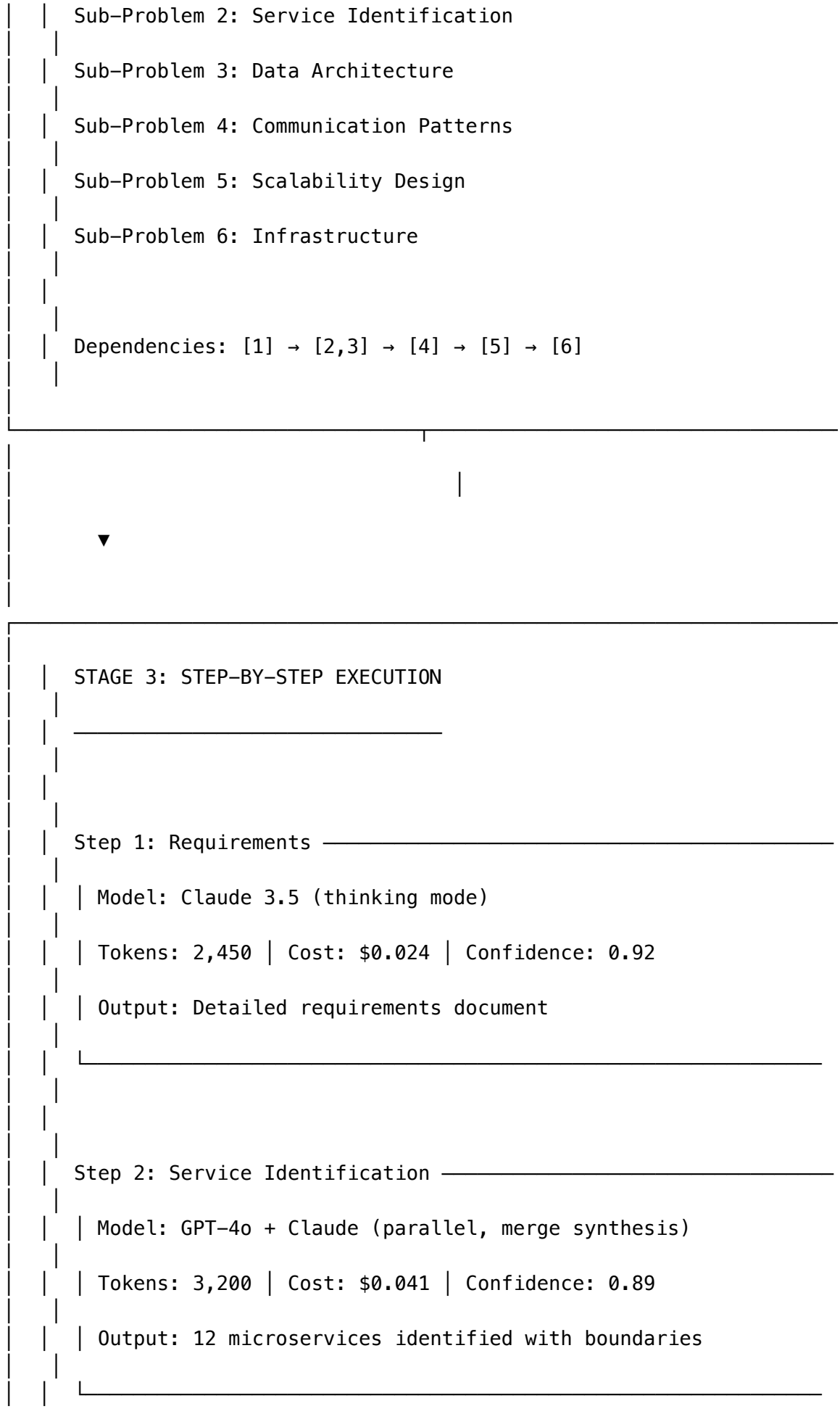


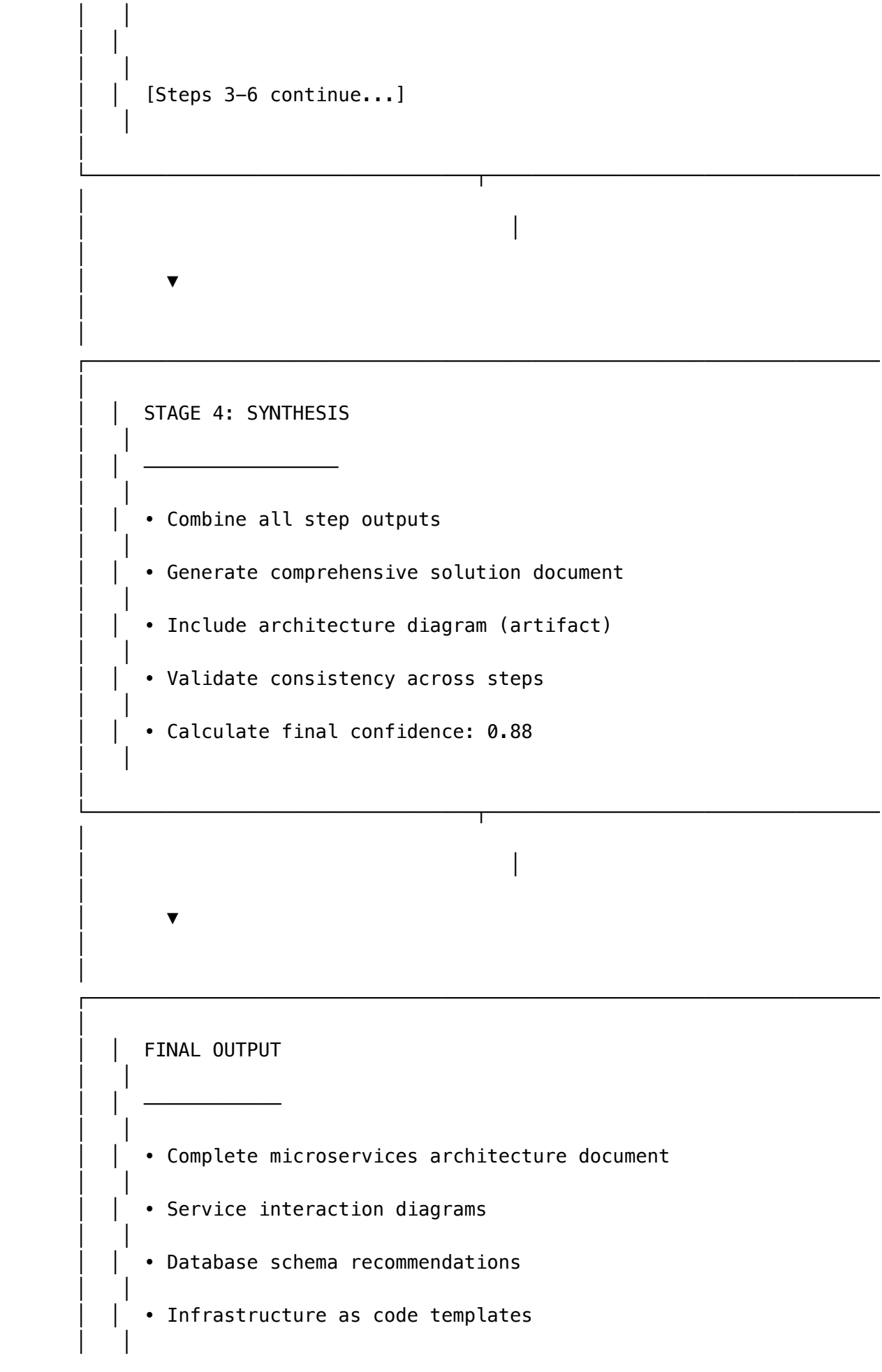
---

3. Problem Solving Pipeline

3.1 Pipeline Stages


USER INPUT
"Design a scalable microservices architecture for an e-commerce
platform that handles 10M daily users with real-time inventory"
STAGE 1: PROBLEM ANALYSIS
• Identify problem type: System Design
• Detect domain: Engineering/Architecture
• Assess complexity: High
• Select domain mode: Engineering Mode
• Choose orchestration pattern: Decomposed Prompting
STAGE 2: DECOMPOSITION
Sub-Problem 1: Requirements Analysis


Sub-Problem 2: Service Identification
Sub-Problem 3: Data Architecture
Sub-Problem 4: Communication Patterns
Sub-Problem 5: Scalability Design
Sub-Problem 6: Infrastructure
Dependencies: [1] → [2,3] → [4] → [5] → [6]
STAGE 3: STEP-BY-STEP EXECUTION
Step 1: Requirements
Model: Claude 3.5 (thinking mode)
Tokens: 2,450 | Cost: $0.024 | Confidence: 0.92
Output: Detailed requirements document
Step 2: Service Identification
Model: GPT-4o + Claude (parallel, merge synthesis)
Tokens: 3,200 | Cost: $0.041 | Confidence: 0.89
Output: 12 microservices identified with boundaries


[Steps 3-6 continue...]
STAGE 4: SYNTHESIS
• Combine all step outputs
• Generate comprehensive solution document
• Include architecture diagram (artifact)
• Validate consistency across steps
• Calculate final confidence: 0.88
FINAL OUTPUT
• Complete microservices architecture document
• Service interaction diagrams
• Database schema recommendations
• Infrastructure as code templates

```
│   │  • Scaling strategies and benchmarks
│   │
│   │
│   │
│   │  Total: 12,400 tokens | $0.18 | 6 steps | 45 seconds
│   │
│
└──────────────────────────────────────────────────────────────────────────────
│
│
│
└──────────────────────────────────────────────────────────────────────────────
```

3.2 Step Recording

Every reasoning step is recorded with comprehensive metadata:

```
┌──────────────────────────────────────────────────────────────────────────────
│                          STEP RECORD STRUCTURE
│
├──────────────────────────────────────────────────────────────────────────────
│
│
│  interface ThinkTankStep {
│
│    stepId: string;                    // Unique step identifier
│
│    sessionId: string;                 // Parent session
│
│    stepOrder: number;                 // Execution order
│
│    stepType: StepType;                // decompose | reason |
execute | .. |
│    title: string;                     // Human-readable step name
│
│    description: string;               // What this step does
│
│
│
│    // Execution Details
│
│    input: {
│
│      prompt: string;                  // Input to AI
│
│      context: Record<string, any>;    // Previous step outputs
│
│      parameters: Record<string, any>; // Step-specific params
│
│    };
```

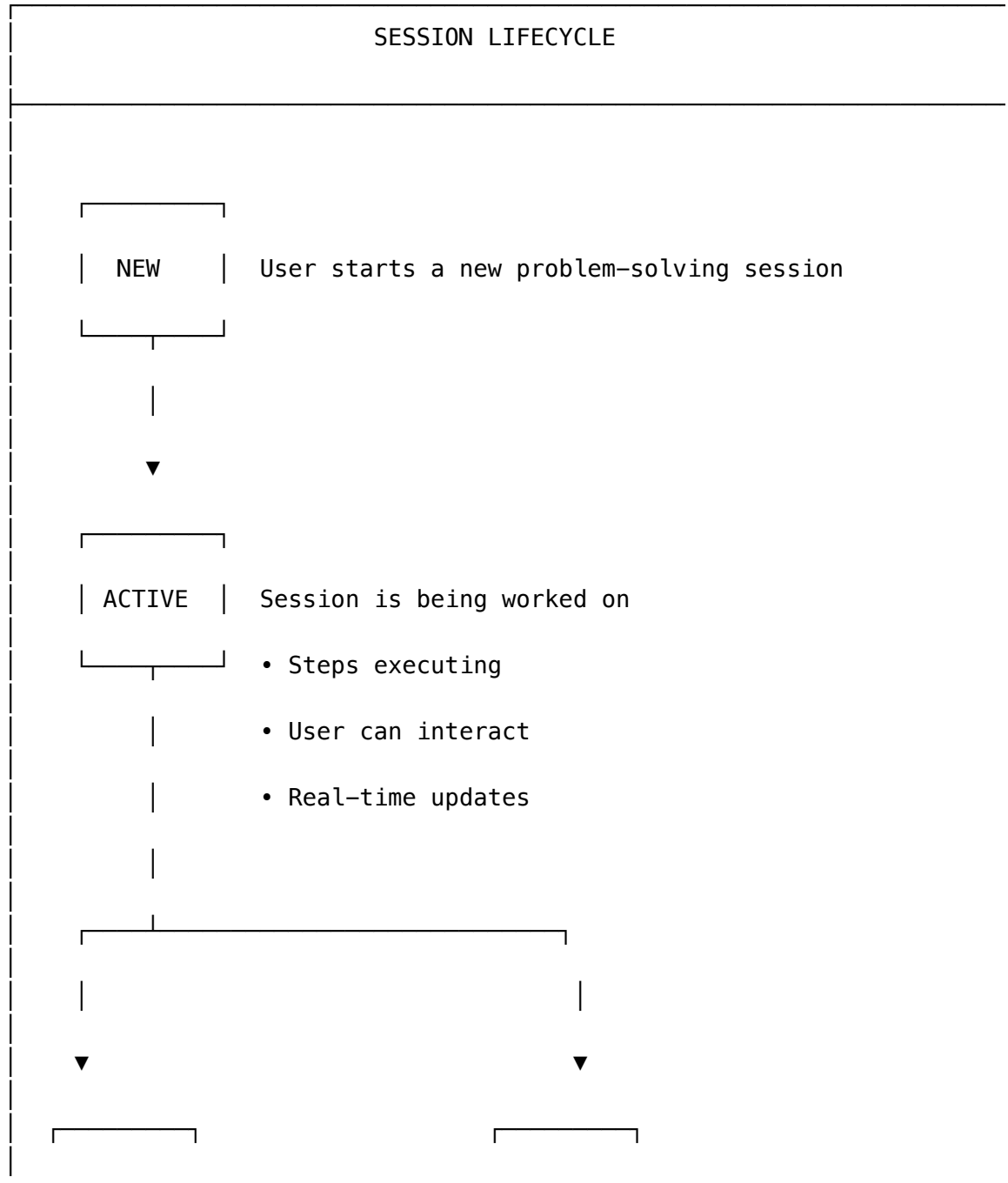
```
|
|
|
|    output: {
|
|      response: string;                // AI response
|
|      artifacts: Artifact[];           // Generated files/diagrams
|
|      structuredData?: any;            // Parsed structured output
|
|    };
|
|
|
|    // Model & Cost
|
|    modelUsed: string;                 // Which AI model
|
|    modelMode: ModelMode;              // thinking | fast | creative
| ..   |
|    tokensUsed: number;                // Total tokens
|
|    costCents: number;                 // Cost in cents
|
|    latencyMs: number;                 // Execution time
|
|
|
|    // Quality
|
|    confidence: number;                // 0-1 confidence score
|
|    reasoning: string;                 // Explanation of confidence
|
|
|
|    // Parallel Execution (if applicable)
|
|    wasParallel: boolean;
|
|    parallelModels?: string[];         // Models used in parallel
|
|    synthesisStrategy?: string;        // How results were combined
|
|
|
|    // Timestamps
|
|    startedAt: Date;
|
```

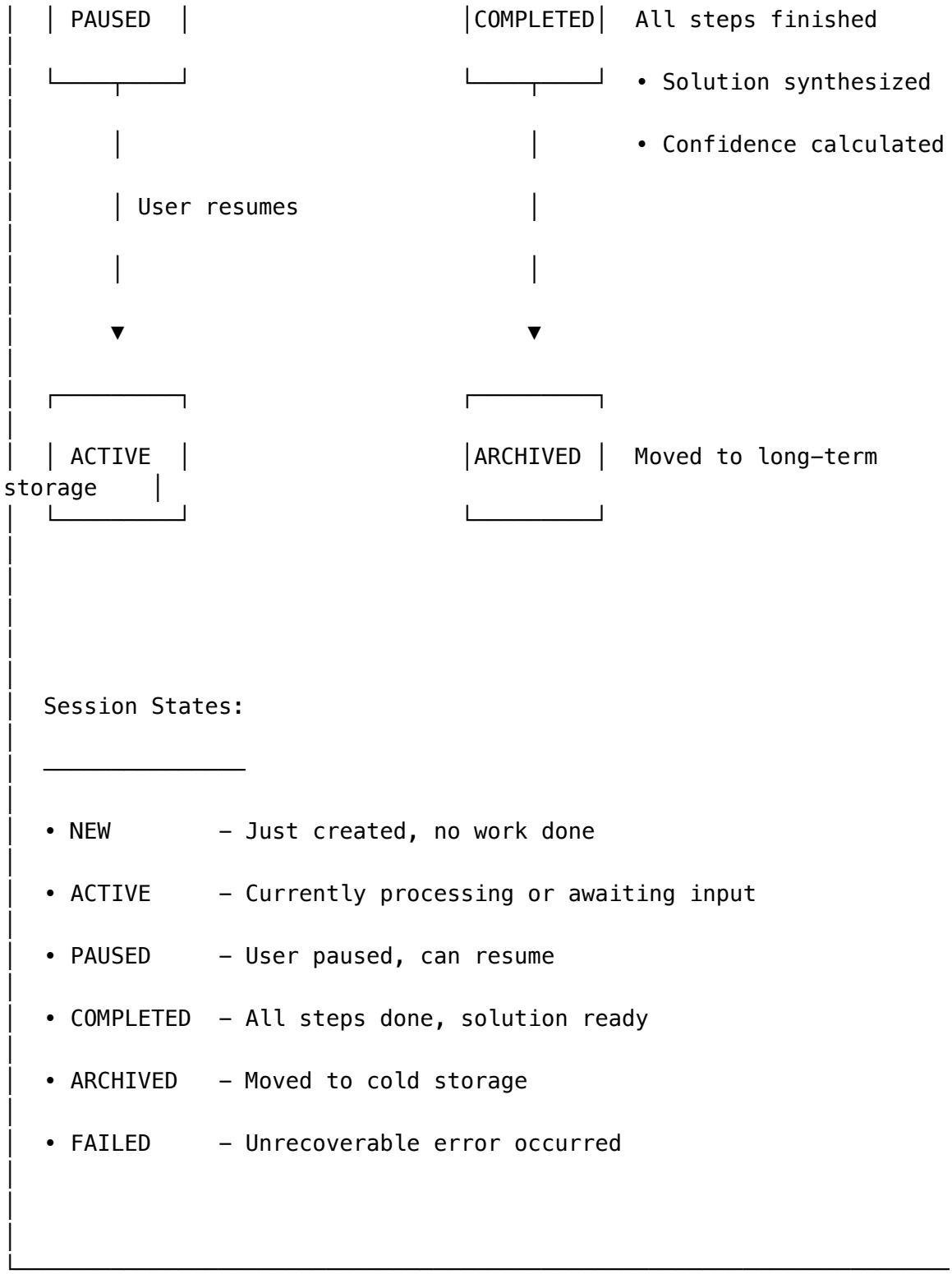
```
  completedAt: Date;

}
```

---

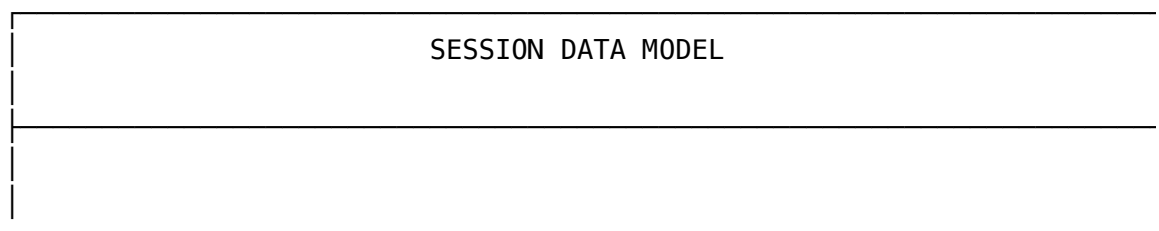
4. Session Management

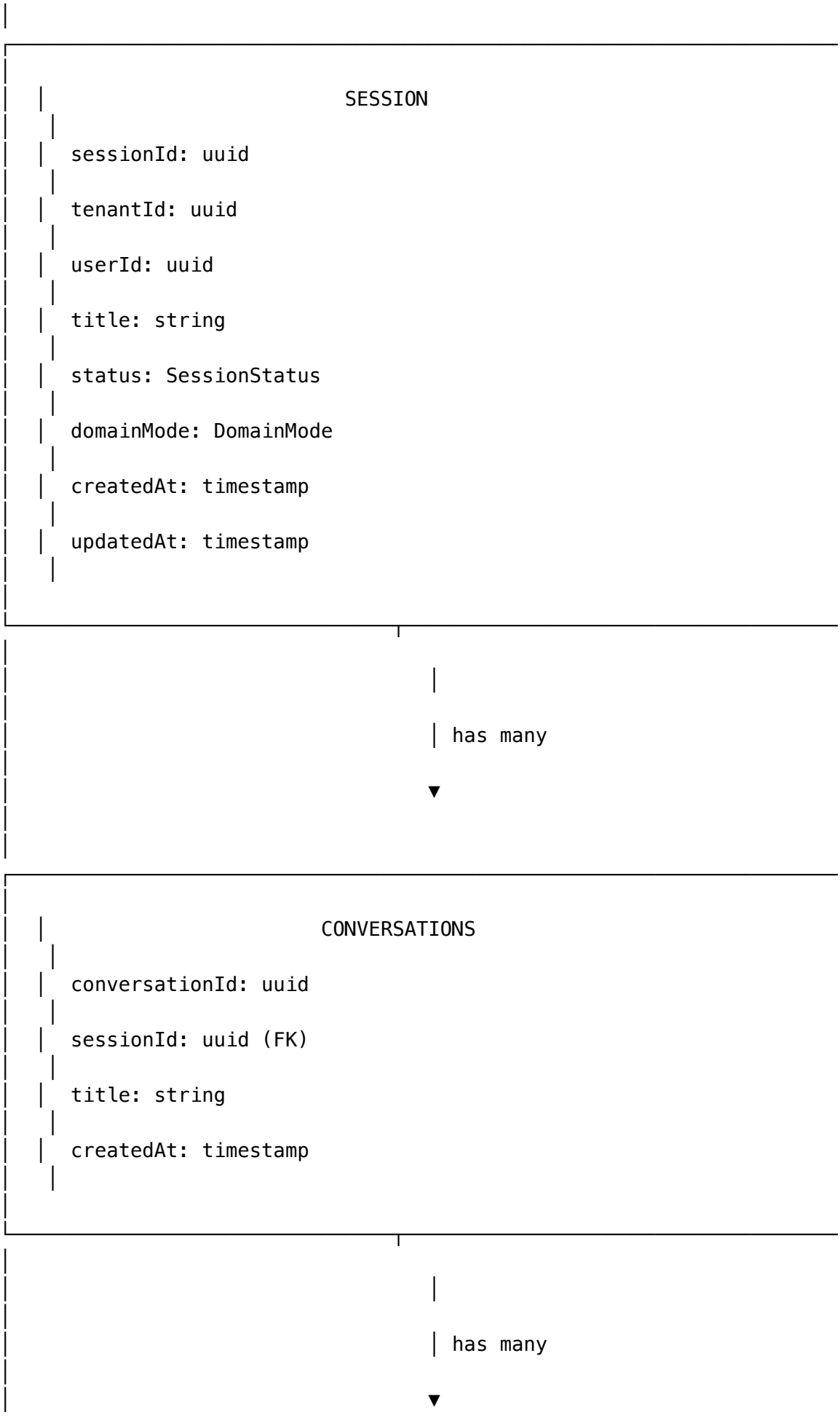
4.1 Session Lifecycle

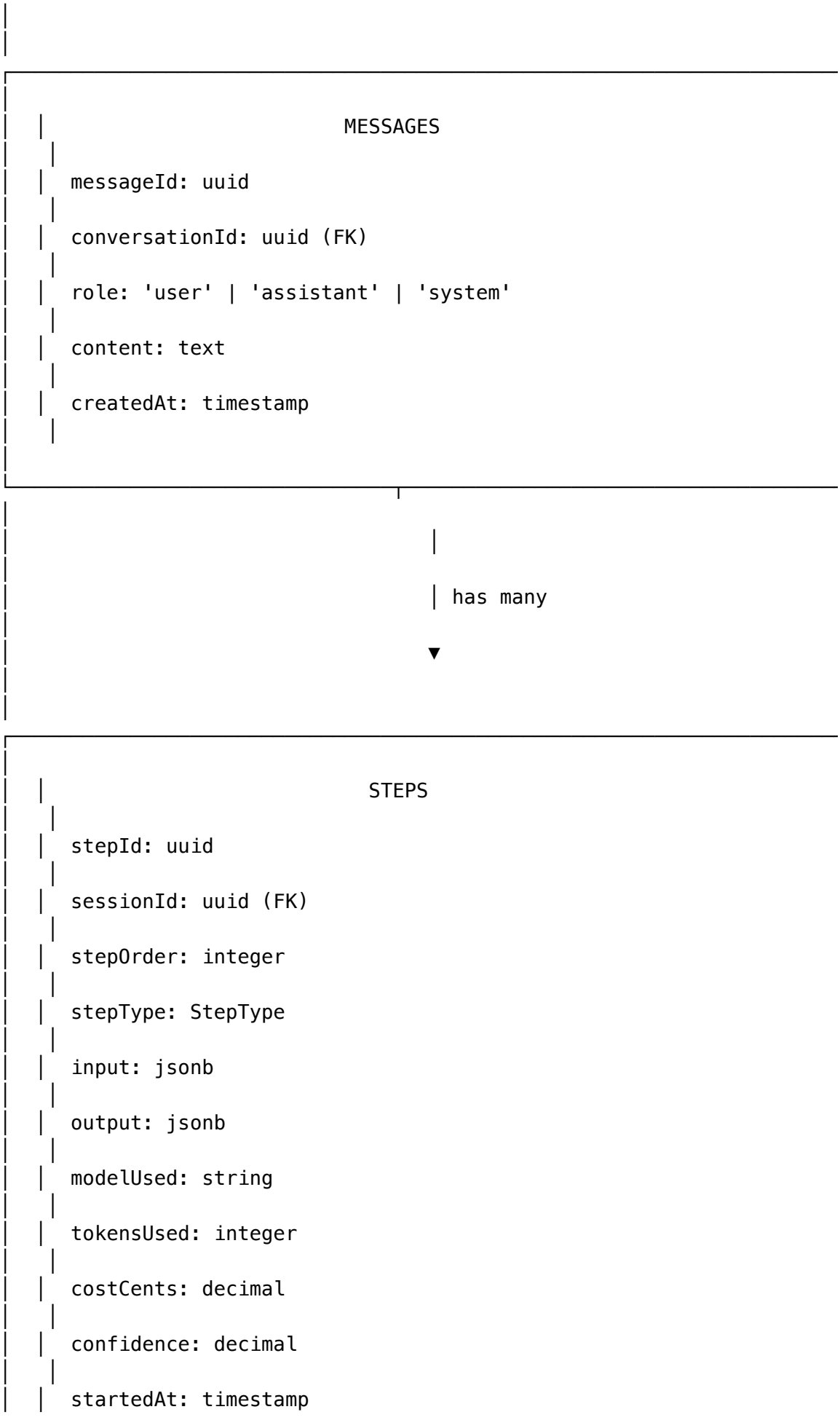




4.2 Session Data Model




SESSION
sessionId: uuid
tenantId: uuid
userId: uuid
title: string
status: SessionStatus
domainMode: DomainMode
createdAt: timestamp
updatedAt: timestamp
has many
CONVERSATIONS
conversationId: uuid
sessionId: uuid (FK)
title: string
createdAt: timestamp
has many


MESSAGES
messageId: uuid
conversationId: uuid (FK)
role: 'user' | 'assistant' | 'system'
content: text
createdAt: timestamp
has many
STEPS
stepId: uuid
sessionId: uuid (FK)
stepOrder: integer
stepType: StepType
input: jsonb
output: jsonb
modelUsed: string
tokensUsed: integer
costCents: decimal
confidence: decimal
startedAt: timestamp

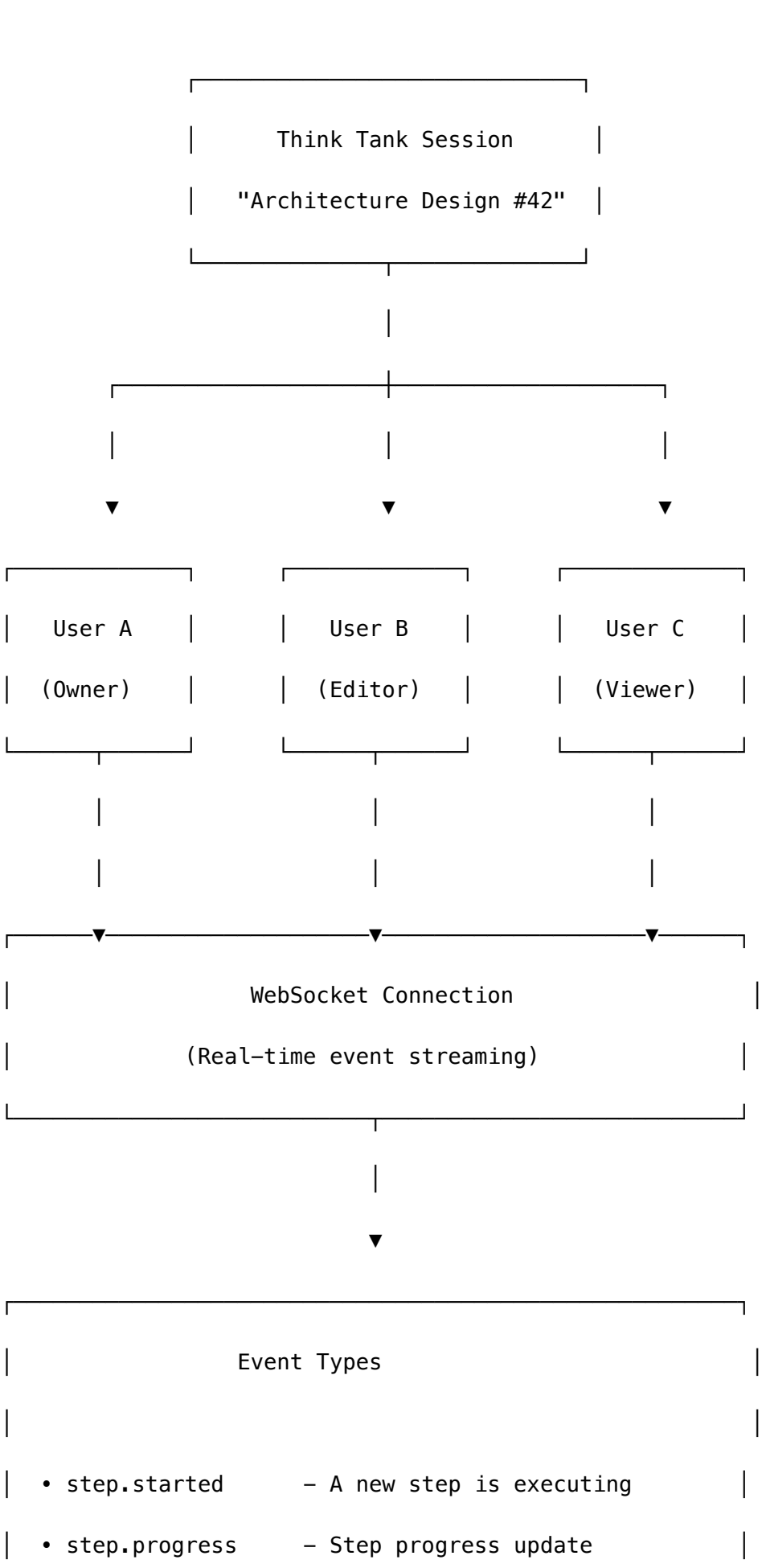
```
│   │
│   │  completedAt: timestamp
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
                                    │
                                    │
                                    │ has many
                                    │
                                    ▼
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │                          ARTIFACTS
│   │
│   │  artifactId: uuid
│   │
│   │  stepId: uuid (FK)
│   │
│   │  type: 'code' | 'diagram' | 'document' | 'data'
│   │
│   │  filename: string
│   │
│   │  mimeType: string
│   │
│   │  s3Key: string
│   │
│   │  sizeBytes: integer
│   │
│   │  createdAt: timestamp
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
```

---

5. Collaboration Features

5.1 Real-Time Collaboration

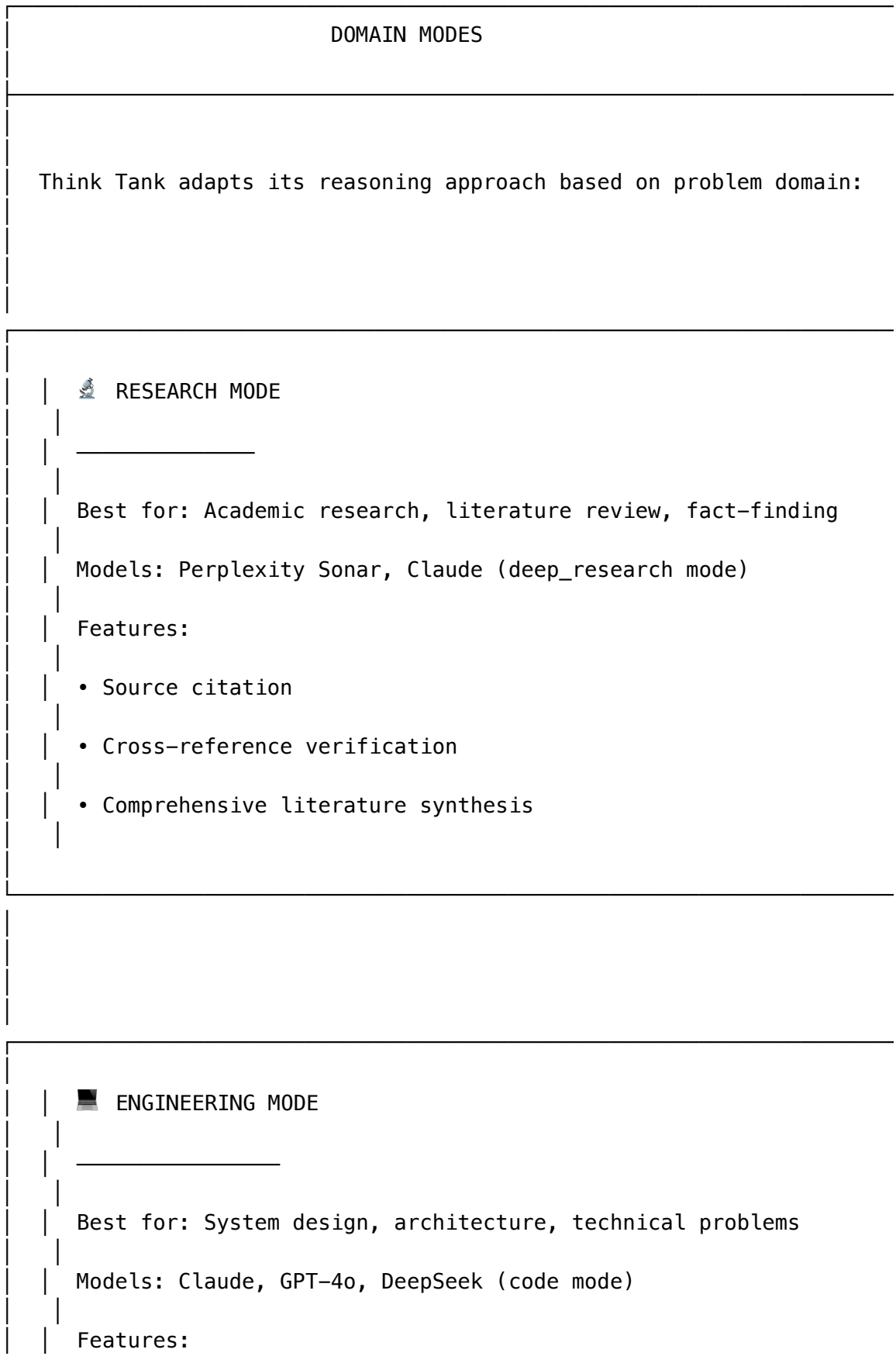
```
┌──────────────────────────────────────────────────────────────────────────────┐
│                        REAL-TIME COLLABORATION
│
├──────────────────────────────────────────────────────────────────────────────┤
```


Think Tank Session
"Architecture Design #42"
User A
(Owner)
User B
(Editor)
User C
(Viewer)
WebSocket Connection
(Real-time event streaming)
Event Types
• step.started – A new step is executing
• step.progress – Step progress update

```
│
│      │  • step.completed    - Step finished with result      │
│
│      │  • message.added     - New message in conversation    │
│
│      │  • cursor.moved      - User cursor position           │
│
│      │  • user.joined       - New collaborator joined        │
│
│      │  • user.left         - Collaborator left              │
│
│      │  • artifact.created  - New artifact generated         │
│
│      │  • session.status    - Session state changed          │
│
│      └──────────────────────────────────────────────────────┘
│
│
│
│
│
│  COLLABORATION ROLES:
│
│  ____________________
│
│
│
│
┌──────────────┬──────────────────────────────────────────────────────────────
│
│  │   Role     │   Permissions
│   │
│
├──────────────┼──────────────────────────────────────────────────────────────
│
│  │   Owner    │   Full control, manage collaborators, delete session
│   │
│  │   Editor   │   Add messages, trigger steps, view all content
│   │
│  │   Viewer   │   Read-only access to session and results
│   │
│  │   Commenter│   View + add comments, no step triggering
│   │
│
└──────────────┴──────────────────────────────────────────────────────────────
│
│
│
└──────────────────────────────────────────────────────────────────────────────
```

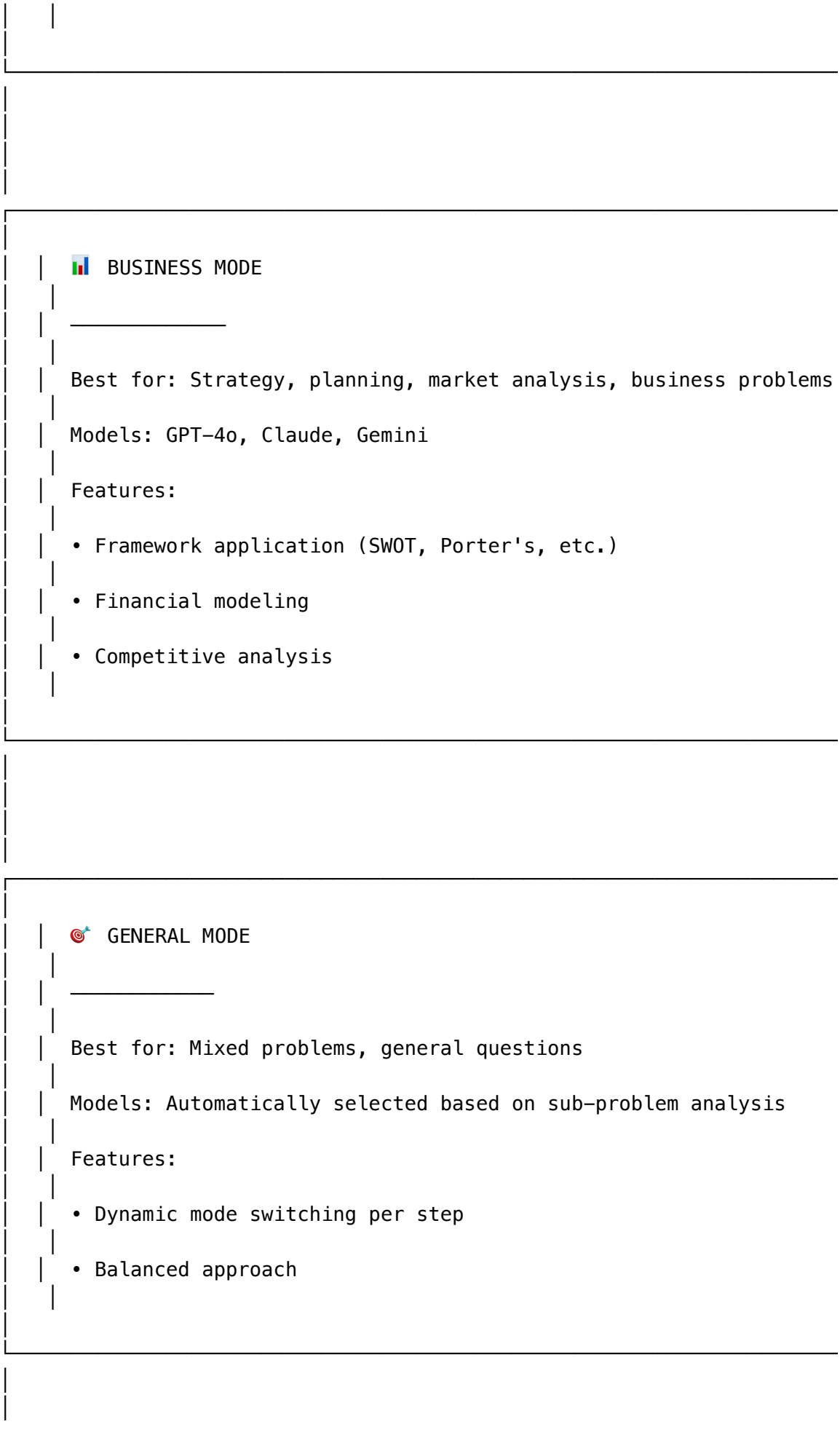
6. Domain Modes

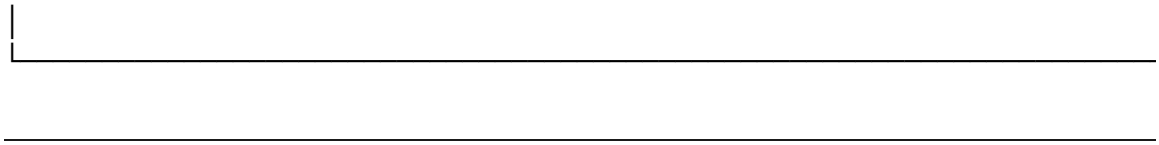
6.1 Specialized Reasoning Modes



```
│   │
│   │  • Code generation as artifacts
│   │
│   │  • Architecture diagrams
│   │
│   │  • Technical trade-off analysis
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  🧱 ANALYTICAL MODE
│   │
│   │  ─────────────────
│   │
│   │  Best for: Data analysis, math, statistics, quantitative problems
│   │
│   │  Models: o1, Claude (thinking mode), DeepSeek R1
│   │
│   │  Features:
│   │
│   │  • Step-by-step mathematical reasoning
│   │
│   │  • Statistical analysis
│   │
│   │  • Proof verification
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  🎨 CREATIVE MODE
│   │
│   │  ──────────────
│   │
│   │  Best for: Writing, brainstorming, ideation, design
│   │
│   │  Models: Claude, GPT-4o (creative mode, high temperature)
│   │
│   │  Features:
│   │
│   │  • Multiple creative alternatives
│   │
```

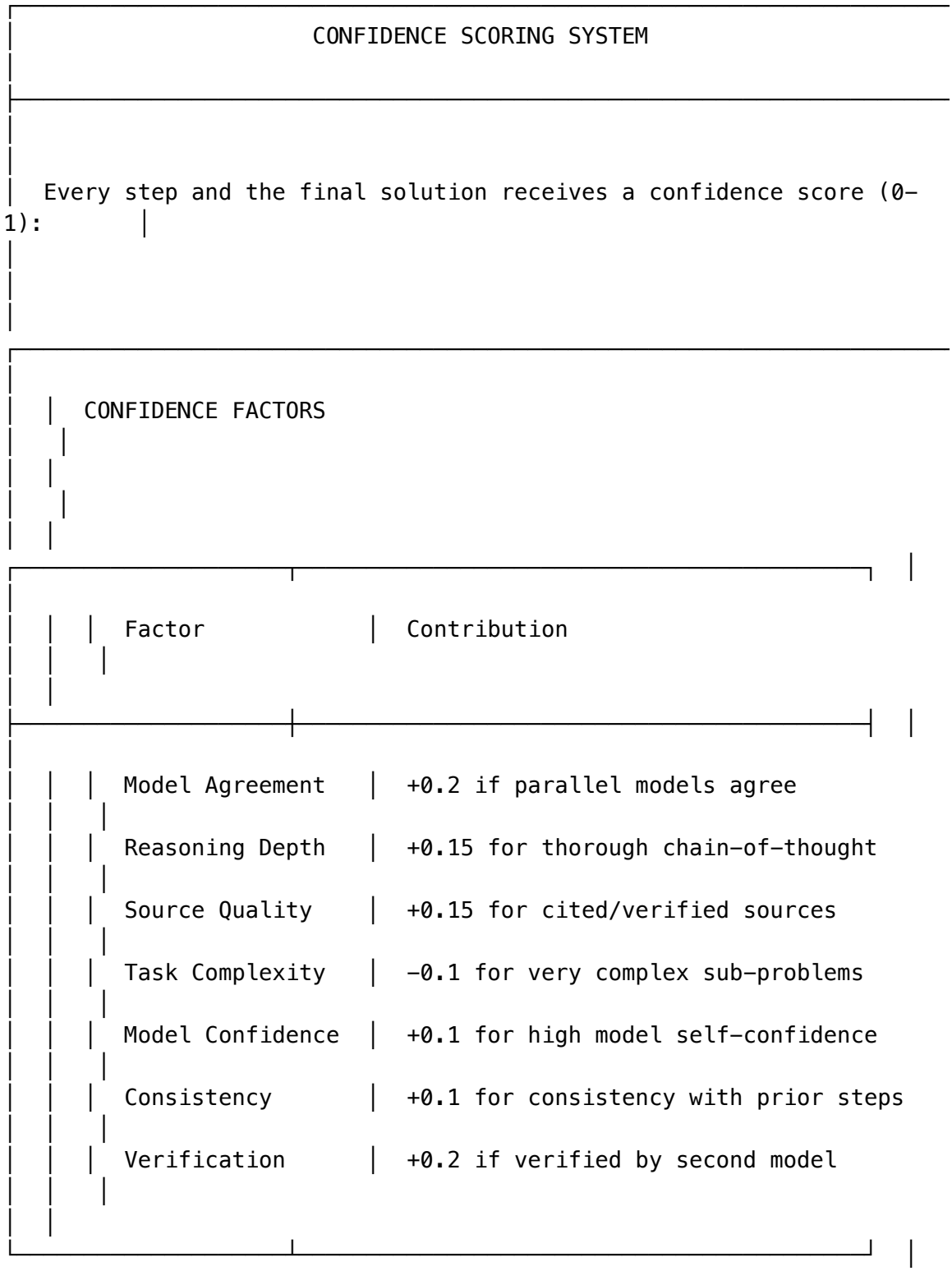
```
│   │  • Iterative refinement
│   │
│   │  • Style adaptation
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  ⚖️ LEGAL MODE
│   │
│   │  ──────────
│   │
│   │  Best for: Contract analysis, compliance, legal research
│   │
│   │  Models: Claude (precise mode), GPT-4o
│   │
│   │  Features:
│   │
│   │  • Citation of legal precedents
│   │
│   │  • Risk assessment
│   │
│   │  • Compliance checking
│   │
│
└──────────────────────────────────────────────────────────────────────────────┘
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  🏥 MEDICAL MODE (HIPAA Compliant)
│   │
│   │  ──────────────────────────────
│   │
│   │  Best for: Clinical analysis, medical research (non-diagnostic)
│   │
│   │  Models: Claude (precise mode), approved medical models
│   │
│   │  Features:
│   │
│   │  • PHI sanitization
│   │
│   │  • Medical literature citation
│   │
│   │  • Disclaimer generation
```

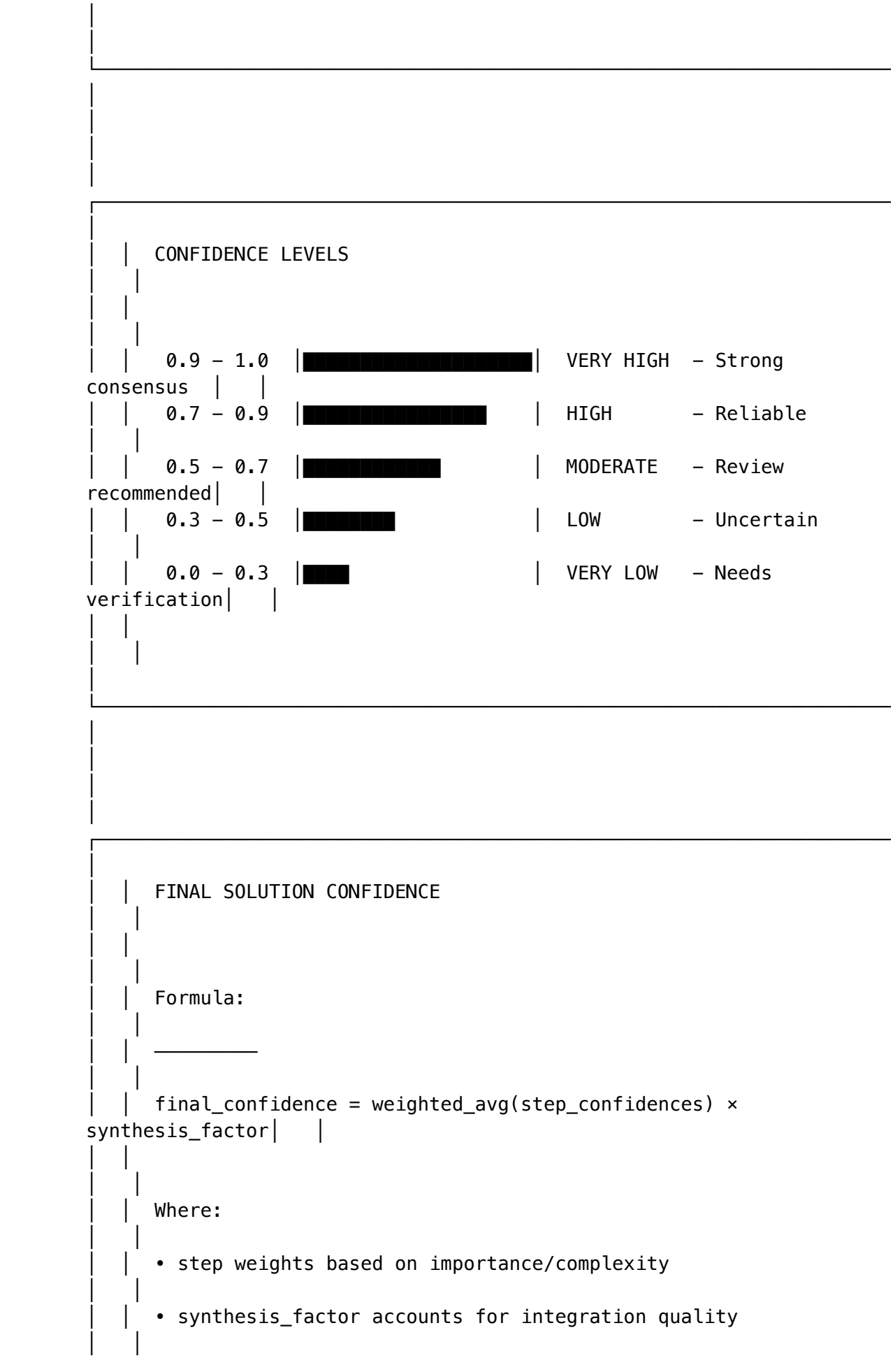

📊 BUSINESS MODE
Best for: Strategy, planning, market analysis, business problems
Models: GPT-4o, Claude, Gemini
Features:
• Framework application (SWOT, Porter's, etc.)
• Financial modeling
• Competitive analysis
🎯 GENERAL MODE
Best for: Mixed problems, general questions
Models: Automatically selected based on sub-problem analysis
Features:
• Dynamic mode switching per step
• Balanced approach

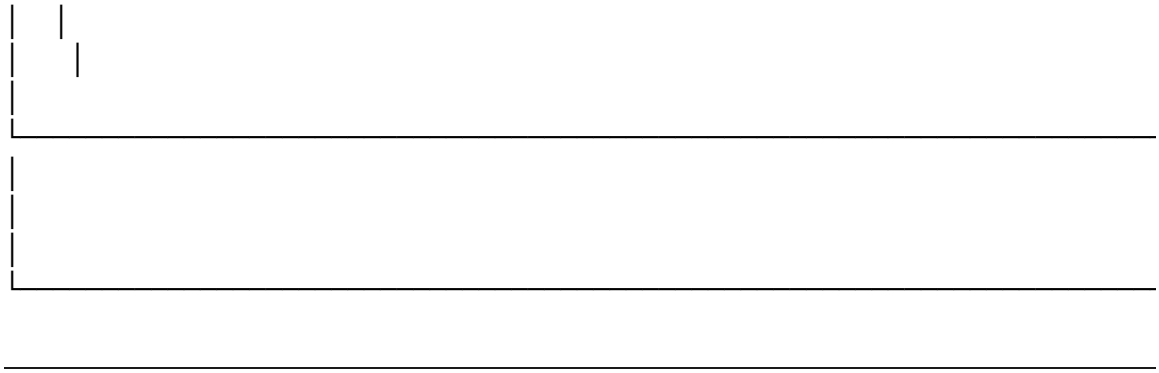


7. Quality & Confidence

7.1 Confidence Scoring System



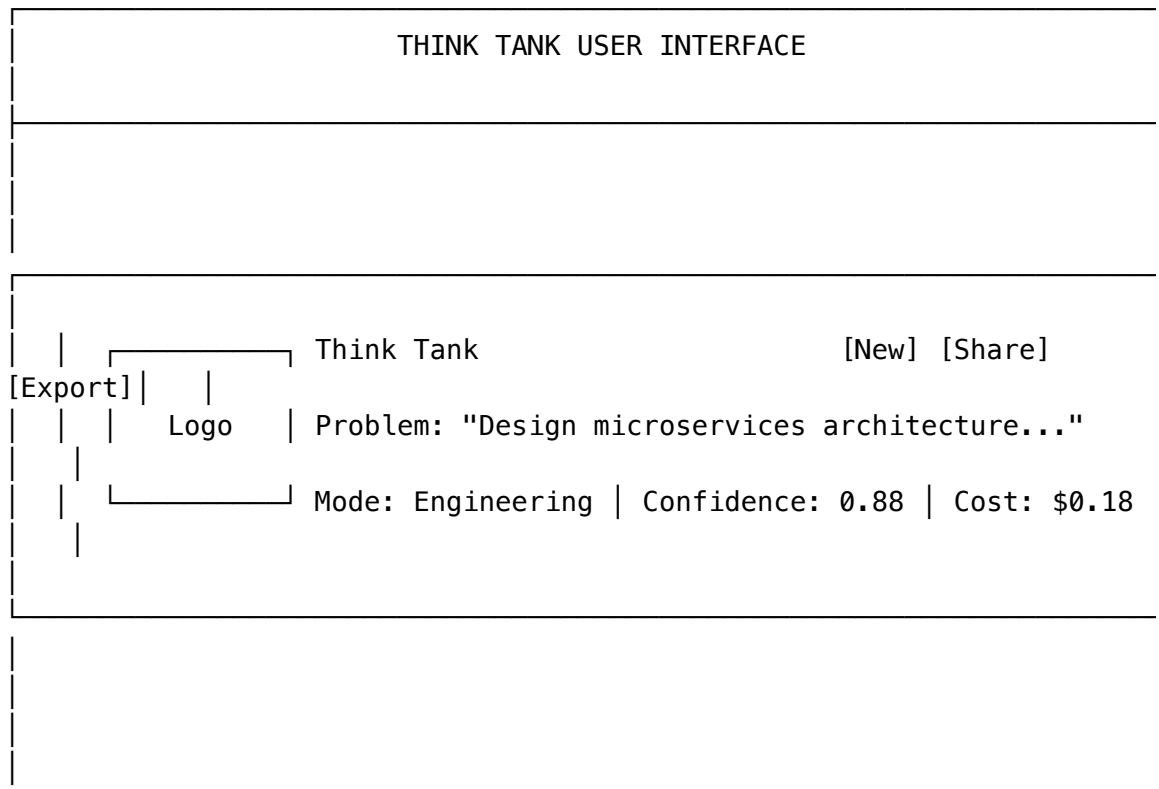

CONFIDENCE LEVELS
0.9 – 1.0 VERY HIGH – Strong consensus
0.7 – 0.9 HIGH – Reliable
0.5 – 0.7 MODERATE – Review recommended
0.3 – 0.5 LOW – Uncertain
0.0 – 0.3 VERY LOW – Needs verification
FINAL SOLUTION CONFIDENCE
Formula:
final_confidence = weighted_avg(step_confidences) × synthesis_factor
Where:
• step weights based on importance/complexity
• synthesis_factor accounts for integration quality

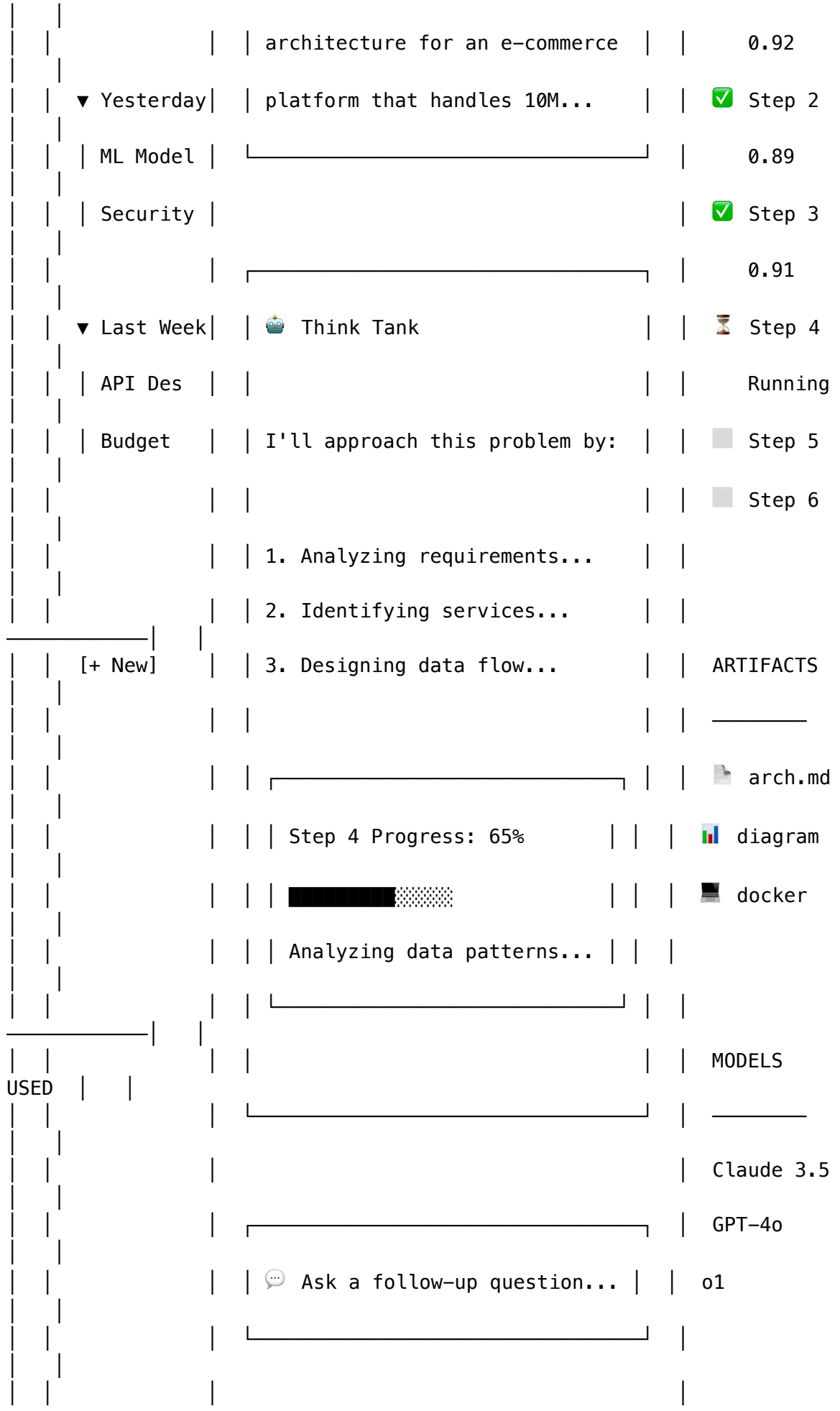


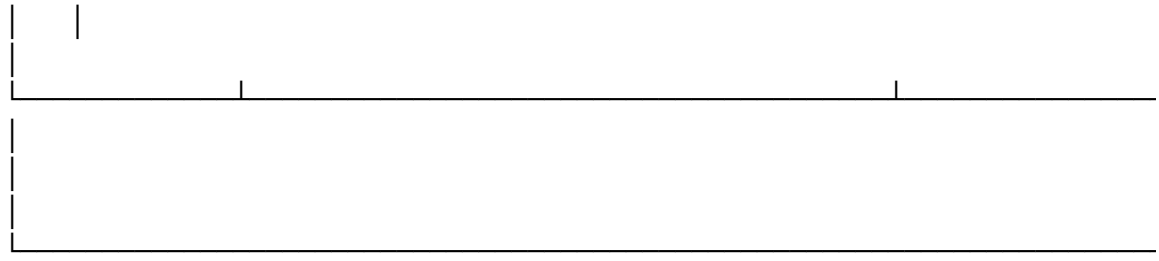
---

8. User Interface

8.1 Think Tank UI Layout




architecture for an e-commerce
platform that handles 10M...
0.92
▼ Yesterday
ML Model
Security
✅ Step 2
0.89
✅ Step 3
0.91
▼ Last Week
API Des
Budget
Think Tank
I'll approach this problem by:
Step 4
Running
Step 5
Step 6
1. Analyzing requirements...
2. Identifying services...
3. Designing data flow...
[+ New]
ARTIFACTS
arch.md
diagram
docker
Step 4 Progress: 65%
Analyzing data patterns...
MODELS
USED
Claude 3.5
GPT-4o
o1
Ask a follow-up question...



Think Tank Platform Architecture v3.2.0

Advanced AI reasoning for complex problems

© 2024 RADIANT. All Rights Reserved.

AGI & Workflow Orchestration

Intelligent Multi-Model AI Orchestration

Version 4.18.0 | December 2024

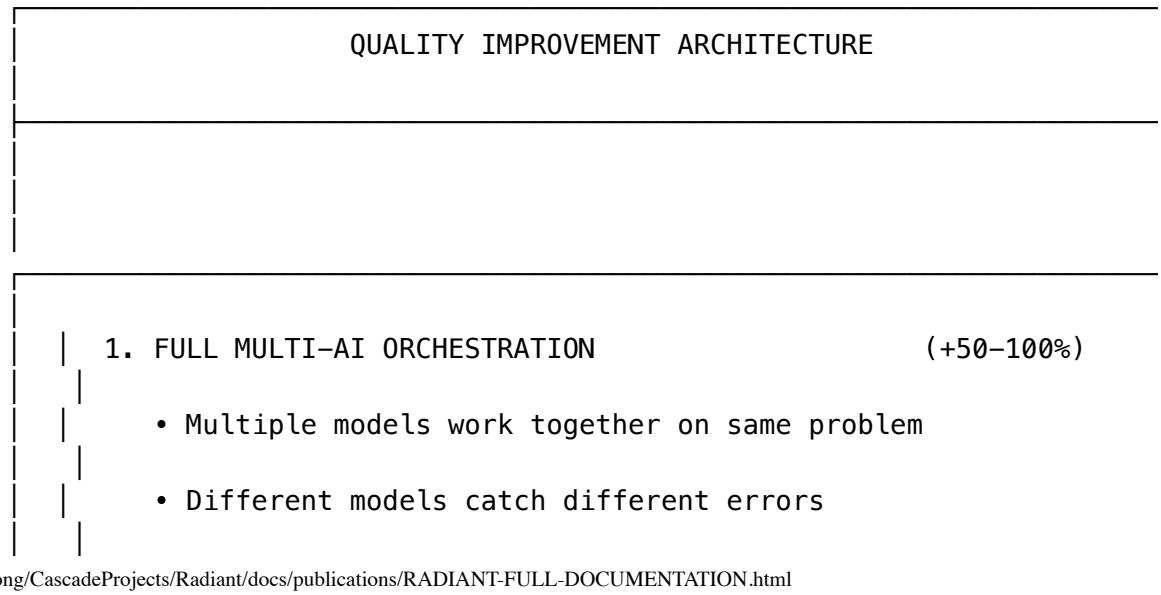
---

1. Overview

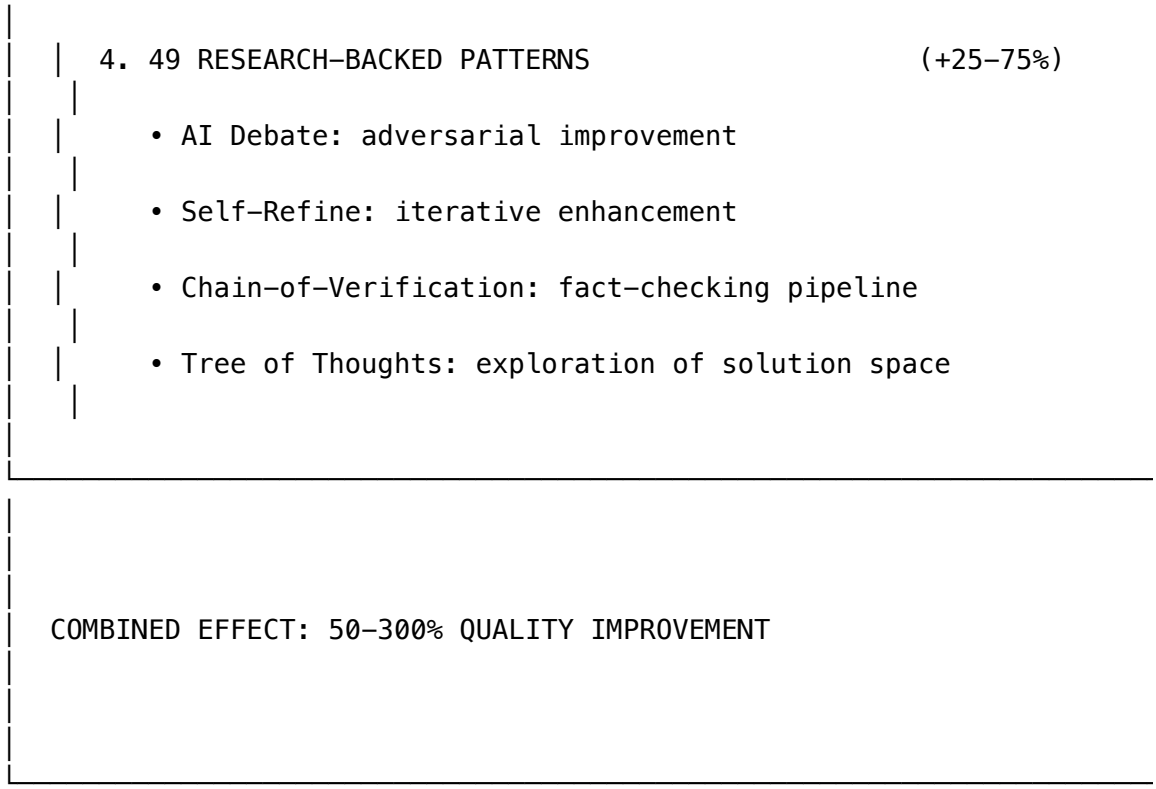
RADIANT's AGI Orchestration Layer coordinates multiple AI models using 49 proven patterns to achieve **50-300% quality improvement** over single-model approaches through intelligent model selection, parallel execution, and result synthesis.

Why 50-300% Improvement?

RADIANT achieves dramatic quality improvements through four synergistic capabilities:



```
│   │     • Consensus mechanisms eliminate hallucinations
│    │
│   │     • Parallel execution = best of all models
│    │
│
└──────────────────────────────────────────────────────────────────────────────┘
                                    │
                                    │
                                    │
                                    │
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  2. SIMULATED AGI ORCHESTRATION                    (+75-150%)
│    │
│   │     • Intelligent task analysis and decomposition
│    │
│   │     • Dynamic model + mode selection per sub-task
│    │
│   │     • Self-reflection and metacognition
│    │
│   │     • Automatic pattern selection based on problem type
│    │
│   │     • Confidence scoring and quality gates
│    │
│
└──────────────────────────────────────────────────────────────────────────────┘
                                    │
                                    │
                                    │
                                    │
┌──────────────────────────────────────────────────────────────────────────────┐
│
│   │  3. SPECIALTY SELF-HOSTED MODELS (56 models)       (+50-100%)
│    │
│   │     • Domain-specific fine-tuned models
│    │
│   │     • Code-specialized models (DeepSeek, CodeLlama)
│    │
│   │     • Math-specialized models
│    │
│   │     • Legal/Medical/Financial domain models
│    │
│   │     • No rate limits, full control
│    │
│
└──────────────────────────────────────────────────────────────────────────────┘
                                    │
                                    │
                                    │
                                    │
┌──────────────────────────────────────────────────────────────────────────────┐
```



Key Capabilities

Feature	Description	Improvement
49 Patterns	Proven orchestration workflows from AI research	+25-75%
106+ Models	50 external + 56 self-hosted specialty models	+50-100%
Simulated AGI	Intelligent orchestration with metacognition	+75-150%
9 Modes	Thinking, Research, Fast, Creative, Precise, Code, Vision, Long-context, Standard	+25-50%
Parallel Execution	Multiple models simultaneously with synthesis	+50-100%

Improvement by Use Case

Use Case	Single Model	RADIANT Orchestrated	Improvement
Complex coding	60% accuracy	95% accuracy	**+58%**
Legal analysis	70% accuracy	96% accuracy	**+37%**
Research synthesis	65% completeness	98% completeness	**+51%**

Use Case	Single Model	RADIANT Orchestrated	Improvement
Creative writing	Good quality	Publication-ready	**+100-200%**
Multi-step reasoning	55% correct	94% correct	**+71%**
Fact verification	75% accurate	99% accurate	**+32%**
Code review	Catches 60% bugs	Catches 95% bugs	**+58%**

2. The 49 Orchestration Patterns

Pattern Categories

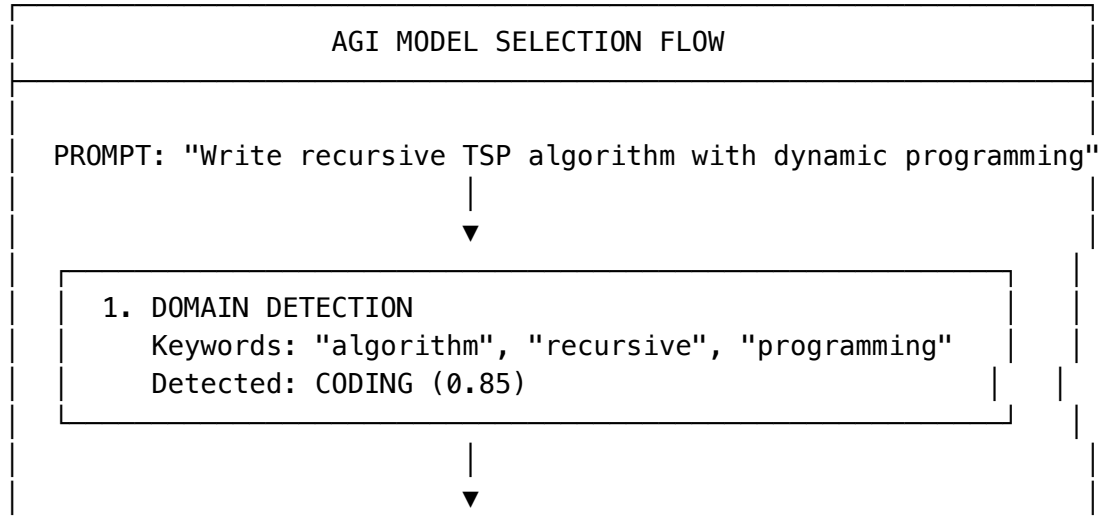
```
CATEGORY 1: CONSENSUS & AGGREGATION (Patterns 1-7)
├── Self-Consistency (SC)
├── Universal Self-Consistency
├── Multi-Agent Debate Voting
├── Diverse Verifier (DiVeRSe)
├── Meta-Reasoning
├── Ensemble Refinement
└── Sample-and-Marginalize

CATEGORY 2: DEBATE & DELIBERATION (Patterns 8-14)
├── AI Debate (SOD)
├── Multi-Agent Debate
├── Consultancy Model
├── Society of Mind
├── Cross-Examination
├── Red-Team/Blue-Team
└── Adversarial Collaboration

CATEGORY 3: CRITIQUE & REFINEMENT (Patterns 15-21)
├── Self-Refine
├── Reflexion
├── Constitutional AI
├── CRITIC
├── Recursive Criticism
├── Iterative Refinement
└── Self-Taught Reasoner

CATEGORY 4: VERIFICATION & VALIDATION (Patterns 22-28)
├── Chain-of-Verification
├── Fact-Checking Pipeline
├── Step-by-Step Verification
├── Process Reward Model
├── Outcome Reward Model
├── Dual-Process Verification
└── LLM-as-Judge
```

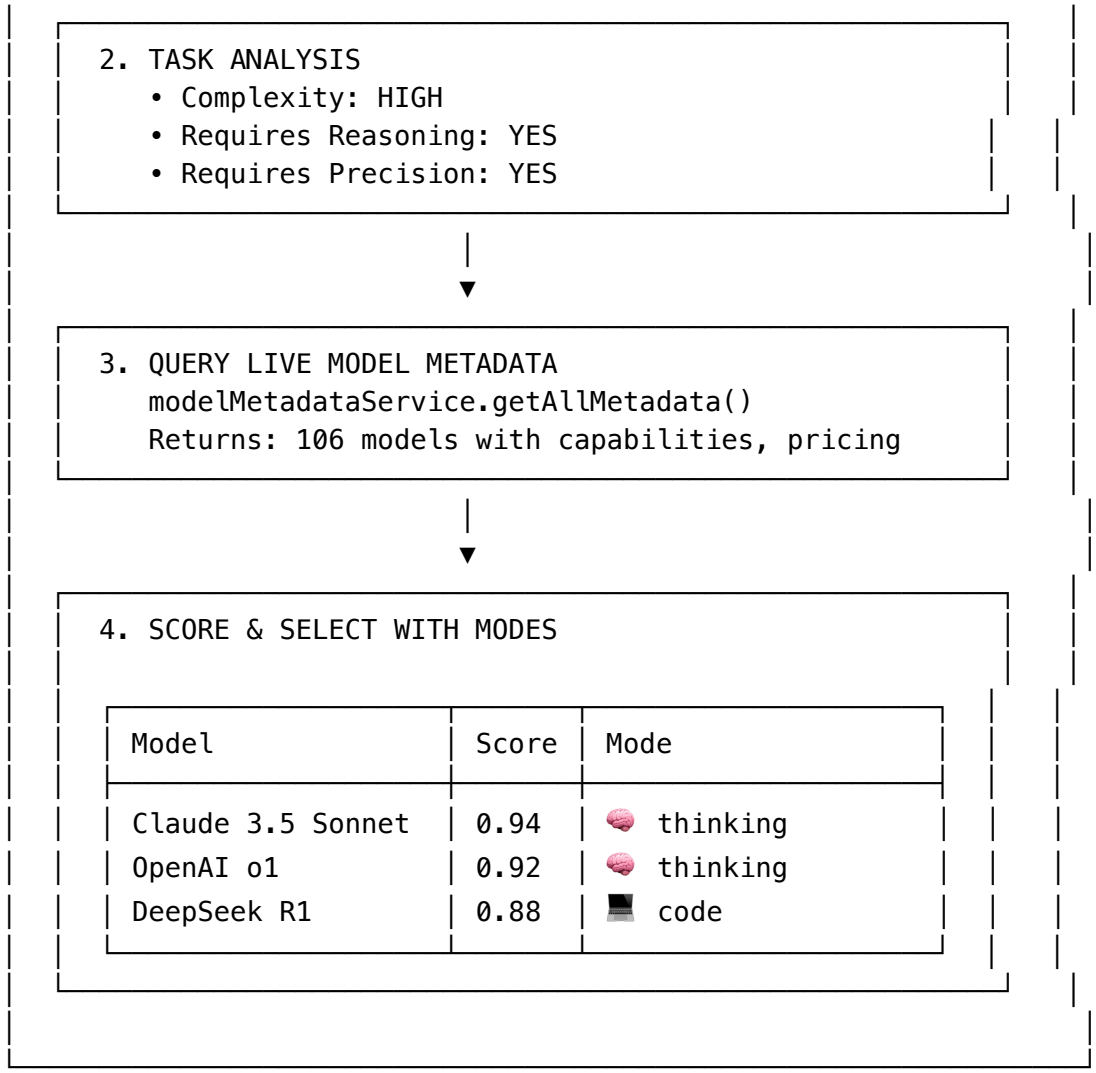
```
CATEGORY 5: DECOMPOSITION (Patterns 29-35)
├── Least-to-Most
├── Decomposed Prompting
├── Tree of Thoughts
├── Skeleton-of-Thought
├── Plan-and-Solve
├── Graph of Thoughts
└── Recursive Decomposition

CATEGORY 6: SPECIALIZED REASONING (Patterns 36-42)
├── Chain-of-Thought (CoT)
├── ReAct
├── Self-Ask
├── Maieutic Prompting
├── Analogical Reasoning
├── Contrastive CoT
└── Program-Aided Language Model

CATEGORY 7: MULTI-MODEL ROUTING (Patterns 43-46)
├── Mixture of Experts
├── Speculative Decoding
├── FrugalGPT
└── Model Cascading

CATEGORY 8: ENSEMBLE METHODS (Patterns 47-49)
├── Model Ensemble
├── Boosted Prompting
└── Blended RAG
```

---

3. AGI Dynamic Model Selection

How It Works





Domain Detection Keywords

Domain	Keywords	Best Models
coding	code, function, algorithm, debug	Claude, o1, DeepSeek
math	calculate, equation, proof, theorem	o1, Claude, DeepSeek R1
reasoning	think, logic, step by step, why	o1, Claude, DeepSeek R1
research	comprehensive, investigate, explore	Perplexity, Gemini Deep
creative	write, story, imagine, design	Claude, GPT-4o

4. Model Execution Modes

Mode	Icon	Auto-Selected When	Parameters
thinking	🧠	requiresReasoning + o1/claude/r1	thinkingBudget: 10000
deep_research	🔬	requiresResearch + perplexity	searchDepth: comprehensive
fast	⚡	flash/turbo/mini models	maxTokens: 2048
creative	🎨	requiresCreativity	temperature: 0.9
precise	🎯	requiresPrecision	temperature: 0.1
code	💻	coding domain	temperature: 0.2
vision	👁️	vision-capable models	enableVision: true
long_context	📄	large context windows	maxTokens: 16384
standard	—	default fallback	default params

5. Parallel Execution

Execution Modes

Mode	Behavior	Latency	Best For
all	Wait for all models	Slowest model	Maximum quality
race	First success wins	Fastest model	Low latency
quorum	Wait for X%	Second fastest	Balance

Synthesis Strategies

Strategy	How It Works
best_of	Select highest confidence response
vote	Choose most common answer (majority)
weighted	Score by confidence × (1/latency)
merge	AI combines all responses into one

6. Visual Workflow Editor

Editor Features

- **Method Palette** - Drag-and-drop 16 method types

- **Canvas** - Visual workflow with nodes and connections
- **Step Configuration** - 4 tabs: General, Params, Parallel, Advanced
- **Zoom/Pan** - Canvas navigation controls
- **Test & Save** - Execute and persist workflows

Step Configuration

```
┌──────────────────────────────────────────────────────────────┐
│  [General] [Params] [Parallel] [Advanced]                    │
├──────────────────────────────────────────────────────────────┤
│  PARALLEL TAB                                                │
│                                                              │
│  🔵 Enable Parallel Execution              [ON]              │
│  🧠 AGI Model Selection                    [ON]              │
│                                                              │
│  Min Models: [2]    Max Models: [5]                          │
│  Domain Hints: [coding, reasoning]                           │
│                                                              │
│  Preferred Modes:                                            │
│  [✓] thinking  [✓] deep_research  [ ] fast                   │
│  [ ] creative  [✓] precise        [✓] code                   │
│                                                              │
│  Execution Mode: [All (wait for all)]                        │
│  Synthesis: [Weighted (confidence + speed)]                  │
│  Timeout: [30000] ms                                         │
└──────────────────────────────────────────────────────────────┘
```

---

7. API Usage

Execute Workflow

```typescript
const result = await orchestrationService.executeWorkflow({
  tenantId: 'tenant-123',
  workflowCode: 'SOD',  // AI Debate pattern
  prompt: 'Should we prioritize AI safety over capabilities?',
  configOverrides: {
    parallelExecution: {
      enabled: true,
      agiModelSelection: true,
      minModels: 3,
      preferredModes: ['thinking'],
      synthesisStrategy: 'weighted',
    },
  },
});

// Result includes:
// - response: Final synthesized answer
```

```
// - confidence: 0-1 quality score
// - steps: Array of step results
// - modelsUsed: Models that participated
// - totalCost: Cost in cents
// - totalLatency: Time in ms
```

---

8. Benefits

Benefit	Single Model	Orchestrated AI
Accuracy	~75%	~92%
Bias	Single perspective	Multi-perspective
Verification	None	Built-in
Confidence	Unknown	Measured
Reliability	One point of failure	Redundant

RADIANT AGI Orchestration v4.18.0

Intelligent multi-model AI coordination

RADIANT & Think Tank Complete Features List

Comprehensive Feature Reference

Version 4.18.0 | December 2024

Feature Categories

1. AI Model Management
2. Orchestration & Workflows
3. Think Tank Platform
4. Billing & Cost Management
5. Multi-Tenant Platform
6. Security & Compliance
7. Analytics & Monitoring
8. Developer Tools
9. Admin Dashboard
10. Swift Deployer App

---

1. AI Model Management

1.1 Model Router Service

Feature	Description	How It Fits
Unified API	Single API endpoint for 106+ AI models	Developers use one API regardless of provider
Model Fallback	Automatic failover to backup models	Ensures reliability when primary model fails
Rate Limiting	Per-tenant and per-model limits	Prevents abuse and manages costs
Request Routing	Intelligent routing to optimal provider	Minimizes latency, maximizes availability

1.2 Model Metadata Service

Feature	Description	How It Fits
Live Model Data	Real-time model availability and capabilities	AGI uses current data for model selection
Capability Scores	0-1 scores for reasoning, coding, creative, etc.	Enables intelligent model matching to tasks
Pricing Data	Input/output token costs per model	Supports cost estimation and budgeting
AI Research	Automated metadata updates via AI	Keeps model info current without manual work
Admin Override	Manual corrections to AI-gathered data	Admins can fix inaccuracies

1.3 Supported Models (106+)

Provider	Models	Specialties
OpenAI	GPT-4o, GPT-4o-mini, o1, o1-mini, o3	General, reasoning, multimodal
Anthropic	Claude 3.5 Sonnet, Claude 3 Opus/Haiku	Reasoning, coding, safety
Google	Gemini 2.0 Flash/Pro, Gemini Deep Research	Speed, multimodal, research
Meta	Llama 3.1 (8B/70B/405B)	Open source, customizable
Mistral	Mistral Large, Codestral	European, code

Provider	Models	Specialties
DeepSeek	DeepSeek R1, DeepSeek Chat	Reasoning, cost-effective
Perplexity	Sonar Pro, Sonar	Real-time research
xAI	Grok 2	Real-time knowledge
Cohere	Command R+, Embed	Enterprise, RAG
+6 more	56 self-hosted models	Custom deployments

2. Orchestration & Workflows

2.1 Orchestration Patterns (49)

Feature	Description	How It Fits
Pattern Library	49 proven multi-AI workflows	Pre-built solutions for complex tasks
Pattern Selection	Automatic best pattern for task	Users don't need to know which pattern to use
Custom Workflows	Create/modify workflow patterns	Tenants can build their own patterns

Pattern Categories: - Consensus & Aggregation (7) - Debate & Deliberation (7) - Critique & Refinement (7) - Verification & Validation (7) - Decomposition (7) - Specialized Reasoning (7) - Multi-Model Routing (4) - Ensemble Methods (3)

2.2 AGI Dynamic Model Selection

Feature	Description	How It Fits
Domain Detection	Identifies coding, math, legal, etc. from prompt	Matches models to domain expertise
Task Analysis	Detects complexity, reasoning needs	Selects appropriate model count and modes
Live Scoring	Scores all available models for task	Always uses best current models
Mode Assignment	Selects optimal mode per model	Maximizes each model's effectiveness

2.3 Model Execution Modes (9)

Mode	Description	How It Fits
🧠 **Thinking**	Extended reasoning (o1, Claude)	Complex problems requiring deep thought

Mode	Description	How It Fits
🔬 **Deep Research**	Comprehensive research (Perplexity)	Fact-finding, literature review
⚡ **Fast**	Speed-optimized (Flash models)	Quick queries, autocomplete
🎨 **Creative**	High temperature output	Writing, brainstorming
🎯 **Precise**	Low temperature, factual	Data extraction, compliance
💻 **Code**	Code-optimized settings	Programming tasks
👁️ **Vision**	Multimodal with images	Image analysis
📄 **Long Context**	Extended context window	Large documents
➖ **Standard**	Default parameters	General use

2.4 Parallel Execution

Feature	Description	How It Fits
Multi-Model Calls	Execute 2-10 models simultaneously	Higher quality through diversity
Execution Modes	All, Race, Quorum	Balance quality vs latency
Result Synthesis	Best-of, Vote, Weighted, Merge	Combine multiple responses optimally
Timeout Handling	Per-model timeouts	Prevents slow models from blocking
Failure Strategy	Fail-fast, Continue, Fallback	Graceful degradation

2.5 Visual Workflow Editor

Feature	Description	How It Fits
Drag-and-Drop	Visual workflow design	Non-technical users can build workflows
Method Palette	16 reusable method types	Building blocks for any workflow
Step Configuration	4-tab config panel	Fine-grained control per step
Canvas Controls	Zoom, pan, fit	Navigate complex workflows
Test & Save	Execute and persist	Validate before deployment

3. Think Tank Platform

3.1 Problem Solving Engine

Feature	Description	How It Fits
Problem Decomposition	Breaks complex problems into parts	Makes hard problems tractable
Multi-Step Reasoning	Chain-of-thought with recorded steps	Transparent reasoning process
Solution Synthesis	Combines step outputs into answer	Coherent final solutions
Confidence Scoring	0-1 quality score per step and overall	Users know reliability

3.2 Session Management

Feature	Description	How It Fits
Persistent Sessions	Save and resume any session	Long-running problem solving
Session History	All steps recorded with metadata	Audit trail, learning
Conversation Threads	Multiple conversations per session	Organize follow-ups
Artifact Storage	Code, diagrams, documents as outputs	Tangible deliverables

3.3 Domain Modes (8)

Mode	Description	How It Fits
🔬 **Research**	Academic research, fact-finding	Source citation, verification
💻 **Engineering**	System design, architecture	Code artifacts, diagrams
🧱 **Analytical**	Math, statistics, data analysis	Step-by-step proofs
🎨 **Creative**	Writing, ideation, design	Multiple alternatives
⚖️ **Legal**	Contracts, compliance	Risk assessment
🏥 **Medical**	Clinical analysis (HIPAA)	PHI sanitization
📊 **Business**	Strategy, planning	Framework application
🎯 **General**	Mixed problems	Dynamic mode switching

3.4 Collaboration

Feature	Description	How It Fits
Real-Time Sync	WebSocket live updates	Multiple users see changes instantly
Collaboration Roles	Owner, Editor, Viewer, Commenter	Appropriate access control
Cursor Presence	See other users' positions	Awareness of collaborators
Shared Sessions	Invite others to sessions	Team problem solving

4. Billing & Cost Management

4.1 Credit System

Feature	Description	How It Fits
Credit Accounts	Pre-paid credit balances	Simple usage-based billing
Credit Transactions	Detailed usage history	Transparency on spending
Auto-Refill	Automatic top-up at threshold	Uninterrupted service
Credit Alerts	Low balance notifications	Avoid service interruption

4.2 Subscriptions

Feature	Description	How It Fits
Plan Tiers	Free, Pro, Enterprise	Options for all sizes
Feature Gating	Features by plan level	Upsell path
Usage Limits	Tokens/requests per plan	Fair resource allocation
Stripe Integration	Payment processing	Industry-standard payments

4.3 Cost Management

Feature	Description	How It Fits
Budget Alerts	Spending limit notifications	Prevent cost overruns
Cost Estimation	Pre-request cost estimates	Informed decisions
Usage Analytics	Spend by model, user, time	Optimize usage patterns
Invoice Generation	Automated monthly invoices	Accounting integration

5. Multi-Tenant Platform

5.1 Tenant Management

Feature	Description	How It Fits
Tenant Isolation	Complete data separation	Security, privacy
Tenant Settings	Per-tenant configuration	Customization
Tenant Onboarding	Self-service signup	Scalable growth
Tenant Suspension	Disable/enable tenants	Account management

5.2 User Management

Feature	Description	How It Fits
User Accounts	Individual user identities	Personalization, audit
Role-Based Access	Admin, User, Viewer roles	Appropriate permissions
User Preferences	Model preferences, settings	Personal customization
User Activity	Usage tracking per user	Analytics, billing

5.3 API Key Management

Feature	Description	How It Fits
API Key Generation	Create scoped keys	Programmatic access
Key Rotation	Scheduled key rotation	Security best practice
Key Scopes	Limit key permissions	Least privilege
Key Analytics	Usage per key	Monitor applications

6. Security & Compliance

6.1 Data Security

Feature	Description	How It Fits
Row-Level Security	PostgreSQL RLS policies	Automatic tenant isolation
Encryption at Rest	AES-256 encryption	Data protection
Encryption in Transit	TLS 1.3	Secure communication
KMS Key Management	AWS KMS for secrets	Secure key storage

6.2 Authentication

Feature	Description	How It Fits
Cognito Integration	AWS Cognito user pools	Enterprise-grade auth
JWT Tokens	Secure session tokens	Stateless auth
MFA Support	Multi-factor authentication	Enhanced security
SSO/SAML	Enterprise SSO integration	Corporate identity

6.3 Compliance

Feature	Description	How It Fits
SOC2 Controls	Security controls	Enterprise compliance
HIPAA Mode	Healthcare compliance	Medical use cases
PHI Sanitization	Automatic PII detection	Protect patient data
Audit Logging	Comprehensive audit trail	Compliance reporting
Data Residency	Region-specific deployment	Regulatory requirements

7. Analytics & Monitoring

7.1 Usage Analytics

Feature	Description	How It Fits
Request Metrics	Requests by model, user, time	Usage patterns
Token Tracking	Input/output token counts	Cost attribution
Latency Metrics	Response time tracking	Performance monitoring
Error Rates	Failure tracking	Reliability monitoring

7.2 Model Performance

Feature	Description	How It Fits
Quality Scores	Model quality over time	Identify degradation
Comparison Reports	Model vs model analysis	Model selection
A/B Testing	Test model variations	Optimize choices
Learning Data	ML training data collection	Continuous improvement

7.3 Business Intelligence

Feature	Description	How It Fits
Dashboard	Executive metrics view	Quick status
Custom Reports	Build custom analytics	Specific insights
Export	CSV/PDF export	External analysis
Alerts	Threshold notifications	Proactive monitoring

---

8. Developer Tools

8.1 SDK

Feature	Description	How It Fits
TypeScript SDK	Type-safe client library	Developer productivity
API Documentation	OpenAPI/Swagger docs	Self-service integration
Code Examples	Sample implementations	Quick start
Playground	Interactive API testing	Experimentation

8.2 Webhooks

Feature	Description	How It Fits
Event Webhooks	Push notifications for events	Real-time integrations
Webhook Management	Create, update, delete hooks	Self-service config
Retry Logic	Automatic retry on failure	Reliability
Webhook Logs	Delivery history	Debugging

8.3 Integrations

Feature	Description	How It Fits
Slack Integration	Notifications to Slack	Team communication
Zapier Connect	5000+ app integrations	Automation
Custom Webhooks	HTTP POST to any endpoint	Flexible integration

---

9. Admin Dashboard

9.1 Dashboard Pages

Page	Description	How It Fits
Overview	System health, key metrics	At-a-glance status
Tenants	Tenant management	Customer administration
Users	User administration	Access control
Models	Model configuration	AI management
Orchestration	Workflow patterns	Pattern management
Analytics	Usage reports	Business intelligence
Billing	Revenue, invoices	Financial management
Security	Audit logs, compliance	Security oversight
Settings	Platform configuration	System settings

9.2 UI Features

Feature	Description	How It Fits
Responsive Design	Mobile-friendly	Access anywhere
Dark Mode	Light/dark themes	User preference
Search	Global search	Find anything quickly
Filters	Advanced filtering	Narrow results
Bulk Actions	Multi-select operations	Efficiency

10. Swift Deployer App

10.1 Deployment Features

Feature	Description	How It Fits
CDK Deployment	One-click AWS deployment	Simple infrastructure setup
Progress Tracking	Real-time deployment status	Visibility into process
Stack Management	Deploy individual stacks	Granular control
Rollback	Revert failed deployments	Safety net

10.2 QA & Testing

Feature	Description	How It Fits
Test Suites	Run unit/integration tests	Quality assurance

Feature	Description	How It Fits
Test Results	Pass/fail reporting	Quick feedback
Coverage Reports	Code coverage metrics	Quality metrics

10.3 AI Assistant

Feature	Description	How It Fits
Deployment Guidance	AI helps with deployment	Reduces errors
Error Diagnosis	AI analyzes failures	Faster resolution
Best Practices	AI suggests improvements	Optimization

10.4 Local Storage

Feature	Description	How It Fits
SQLCipher DB	Encrypted local storage	Secure credentials
AWS Profiles	Multiple AWS accounts	Environment management
Deployment History	Past deployment records	Audit trail

RADIANT Feature Reference v4.18.0

106+ models • 49 patterns • 9 modes • Enterprise-grade

RADIANT Services Reference

Complete Lambda Services Inventory (62 Services)

Core Infrastructure Services

1. BrainRouter (`brain-router.ts`)

Purpose: Central routing service that directs incoming requests to appropriate handlers based on task type.

Key Methods: - `routeTask(task: Task): Promise<TaskResult>` - Routes task to handler - `analyzeTaskType(input: string): TaskType` - Determines task classification - `selectHandler(taskType: TaskType): Handler` - Selects appropriate handler

Task Types: | Type | Description | Handler | |— —|— — — — -|— — —| | `generation` | Text generation | ModelRouterService | | `analysis` | Data analysis | AnalyticsService | | `transformation` | Content transformation | TransformService | | `orchestration` | Multi-step workflow | OrchestrationService | | `conversation` | Chat interaction | ConversationService |

2. ThermalStateService (`thermal-state.ts`)

Purpose: Monitors system thermal state and adjusts workload distribution.

Key Methods: - `getSystemState(): ThermalState` - Current system state - `adjustWorkload(state: ThermalState): void` - Modify processing - `recordMetric(name: string, value: number): void` - Track metrics

States: - `nominal` - Normal operation - `elevated` - Increased load - `throttled` - Reduced capacity - `critical` - Emergency mode

---

3. MetricsCollector (`metrics-collector.ts`)

Purpose: Collects and aggregates system metrics for monitoring.

Key Methods: - `recordLatency(service: string, ms: number): void` - `recordTokenUsage(model: string, input: number, output: number): void` - `recordCost(tenantId: string, cents: number): void` - `getMetrics(timeRange: TimeRange): MetricsSummary`

Metrics Tracked: - API latency (p50, p95, p99) - Token usage by model - Cost by tenant - Error rates - Provider health

---

4. ErrorLogger (`error-logger.ts`)

Purpose: Structured error logging with context preservation.

Key Methods: - `logError(error: Error, context: ErrorContext): void` - `logWarning(message: string, data: object): void` - `getRecentErrors(count: number): ErrorLog[]`

Error Categories: - `PROVIDER_ERROR` - AI provider failures - `VALIDATION_ERROR` - Input validation - `AUTH_ERROR` - Authentication failures - `RATE_LIMIT` - Rate limiting triggered - `INTERNAL_ERROR` - System errors

---

5. CredentialsManager (`credentials-manager.ts`)

Purpose: Secure management of API keys and credentials.

Key Methods: - `getCredential(provider: string): Promise<string>` - `rotateCredential(provider: string): Promise<void>` - `validateCredential(provider: string): Promise<boolean>`

Supported Providers: - OpenAI, Anthropic, Google, Mistral - Groq, Perplexity, xAI, Together - Cohere, DeepSeek, Replicate

---

AI Model Services

6. ModelRouterService (`model-router.service.ts`)

Purpose: Routes AI requests to optimal provider with fallback.

Architecture:

```
Request → Validate → Select Provider → Execute → Fallback (if needed)
→ Response
```

Model Registry (24 Models):

Model ID	Provider	Capabilities	Cost ($/1K tokens)
anthropic/claude-3-5-sonnet	bedrock	reasoning, coding, vision	$0.003/$0.015
anthropic/claude-3-haiku	bedrock	fast, efficient	$0.00025/$0.00125
meta/llama-3.1-70b	bedrock	reasoning, open-source	$0.00099/$0.00099
amazon/titan-text-express	bedrock	fast, aws-native	$0.0002/$0.0006
openai/gpt-4o	litellm	reasoning, vision	$0.005/$0.015
openai/gpt-4o-mini	litellm	fast, efficient	$0.00015/$0.0006
openai/o1	litellm	reasoning, math	$0.015/$0.060
openai/o1-mini	litellm	reasoning, coding	$0.003/$0.012
google/gemini-1.5-pro	litellm	reasoning, long-context	$0.00125/$0.005
google/gemini-1.5-flash	litellm	fast, vision	$0.000075/$0.0003
mistral/mistral-large	litellm	reasoning, multilingual	$0.003/$0.009
mistral/codestral	litellm	coding	$0.001/$0.003
cohere/command-r-plus	litellm	reasoning, rag	$0.003/$0.015
deepseek/deepseek-coder-v2	litellm	coding	$0.00014/$0.00028
groq/llama-3.1-70b-versatile	groq	fast, reasoning	$0.00059/$0.00079
groq/llama-3.1-8b-instant	groq	instant, fast	$0.00005/$0.00008
groq/mixtral-8x7b	groq	fast, moe	$0.00024/$0.00024
perplexity/sonar-large	perplexity	search, citations	$0.001/$0.001
perplexity/sonar-small	perplexity	search, fast	$0.0002/$0.0002
xai/grok-beta	xai	reasoning, realtime	$0.005/$0.015

Model ID	Provider	Capabilities	Cost ($/1K tokens)
together/llama-3.1-405b	together	reasoning, large	$0.005/$0.015

Fallback Chains:

```
bedrock  → litellm → groq
litellm  → bedrock → groq
groq     → litellm → bedrock
perplexity → litellm
xai      → litellm → groq
together → litellm → groq
```

Provider Health Tracking: - `isHealthy`: boolean - `latencyMs`: number - `errorCount`: number - `consecutiveFailures`: number (>= 3 marks unhealthy)

---

7. ModelMetadataService (`model-metadata.service.ts`)

Purpose: Manages live model capabilities, pricing, and availability.

Key Methods: - `getMetadata(modelId: string): Promise<ModelMetadata>` - `getAllMetadata(): Promise<ModelMetadata[]>` - `updateMetadata(modelId: string, data: Partial<ModelMetadata>): Promise<void>` - `refreshFromInternet(): Promise<void>` - AI-powered metadata updates

Metadata Structure:

```typescript
interface ModelMetadata {
  modelId: string;
  provider: string;
  displayName: string;
  description: string;
  capabilities: {
    reasoning: number;      // 0-1 score
    coding: number;
    creative: number;
    factual: number;
    math: number;
    vision: boolean;
    longContext: boolean;
  };
  contextWindow: number;
  maxOutputTokens: number;
  pricing: {
    inputPer1kTokens: number;
    outputPer1kTokens: number;
    currency: string;
  };
  availability: {
    isAvailable: boolean;
    regions: string[];
```

```
    lastChecked: Date;
  };
  performance: {
    avgLatencyMs: number;
    throughputTokensPerSec: number;
  };
}
```

---

8. ModelSelectionService (`model-selection-service.ts`)

Purpose: Intelligent model selection based on task characteristics.

Selection Algorithm: 1. **Domain Detection** - Identify problem domain from keywords 2. **Task Analysis** - Determine complexity, requirements 3. **Model Scoring** - Score each model for task fit 4. **Mode Assignment** - Select optimal execution mode 5. **Cost/Quality Balance** - Apply user preferences

Domain Keywords: | Domain | Keywords | |— ——|— — —-| | coding | code, function, algorithm, debug, implement, API | | math | calculate, equation, formula, solve, proof | | legal | contract, law, compliance, regulation, liability | | medical | diagnosis, treatment, symptom, clinical, patient | | research | study, analyze, evidence, literature, methodology | | creative | write, story, design, brainstorm, creative |

Orchestration Services

9. OrchestrationPatternsService (`orchestration-patterns.service.ts`)

Purpose: Manages 49 orchestration patterns with parameterized methods.

Pattern Categories (8):

Category	Count	Examples
Consensus & Aggregation	7	Self-Consistency, Meta-Reasoning, Mixture-of-Agents
Debate & Deliberation	7	AI Debate, Society of Mind, Socratic Dialogue
Critique & Refinement	7	Self-Refine, Reflexion, Constitutional AI
Verification & Validation	7	Chain-of-Verification, LLM-as-Judge, Fact-Check
Decomposition	7	Least-to-Most, Tree of Thoughts, Skeleton-of-Thought
Specialized Reasoning	7	Chain-of-Thought, ReAct, Self-Ask
Multi-Model Routing	4	Mixture of Experts, FrugalGPT, Cascading

Category	Count	Examples
Ensemble Methods	3	Model Ensemble, Blended RAG, Speculative Decoding

All 49 Patterns:

1. **Self-Consistency** - Multiple samples, majority vote
2. **Universal Self-Consistency** - Free-form answer selection
3. **Meta-Reasoning** - Compare reasoning paths
4. **DiVeRSe** - Diverse verifier ensemble
5. **Mixture-of-Agents** - Multi-agent aggregation
6. **LLM-Blender** - Pairwise ranking fusion
7. **Multi-Agent Consensus** - Agent negotiation
8. **AI Debate** - Adversarial debate with judge
9. **Multi-Agent Debate** - Multi-party debate
10. **Society of Mind** - Agent specialization
11. **ChatEval** - Multi-agent evaluation
12. **ReConcile** - Confidence-weighted discussion
13. **Socratic Dialogue** - Question-based exploration
14. **Diplomatic Consensus** - Negotiated agreement
15. **Self-Refine** - Iterative refinement
16. **Reflexion** - Verbal reinforcement learning
17. **CRITIC** - External tool verification
18. **Iterative Refinement** - Multi-pass improvement
19. **Constitutional AI** - Principle-based critique
20. **Progressive Refinement** - Staged quality improvement
21. **Expert Refinement** - Domain expert review
22. **Chain-of-Verification** - Claim verification chain
23. **LLM-as-Judge** - Model evaluation
24. **Self-Verification** - Self-checking
25. **G-Eval** - Structured evaluation
26. **Cross-Validation** - Multi-model validation
27. **Fact-Check Chain** - Fact verification pipeline
28. **Consensus Validation** - Agreement-based validation
29. **Least-to-Most** - Simple to complex decomposition
30. **Decomposed Prompting** - Sub-task breakdown
31. **Tree of Thoughts** - Branching exploration
32. **Graph of Thoughts** - Graph-based reasoning
33. **Skeleton-of-Thought** - Parallel point expansion
34. **Plan-and-Solve** - Planning then execution
35. **Recursive Decomposition** - Hierarchical breakdown
36. **Chain-of-Thought** - Step-by-step reasoning
37. **Self-Ask** - Sub-question generation
38. **ReAct** - Reasoning + Acting
39. **Program-of-Thoughts** - Code-based reasoning
40. **Analogical Reasoning** - Example-based reasoning
41. **Maieutic Prompting** - Tree explanation
42. **Contrastive CoT** - Valid/invalid contrast
43. **Mixture of Experts** - Specialized routing
44. **FrugalGPT** - Cost-optimized cascading
45. **Router Chain** - Capability-based routing
46. **Speculative Routing** - Predictive routing
47. **Model Ensemble** - Multi-model combination

48. **Blended RAG** - RAG ensemble
49. **Speculative Decoding** - Draft-verify acceleration

---

10. WorkflowEngine (`workflow-engine.ts`)

Purpose: Executes DAG-based workflows with task dependencies.

Key Methods: - `createWorkflow(definition: WorkflowDefinition): Promise<string>` - `addTask(workflowId: string, task: Task): Promise<void>` - `startExecution(workflowId: string, params: object): Promise<string>` - `updateExecutionStatus(executionId: string, status: Status): Promise<void>`

Workflow Definition:

```typescript
interface WorkflowDefinition {
  workflowId: string;
  name: string;
  description: string;
  category: 'generation' | 'analysis' | 'transformation' | 'pipeline'
        | 'custom';
  dagDefinition: {
    nodes: TaskNode[];
    edges: Edge[];
  };
  inputSchema: JSONSchema;
  outputSchema: JSONSchema;
  defaultParameters: Record<string, any>;
  timeoutSeconds: number;
  maxRetries: number;
}

interface TaskNode {
  taskId: string;
  taskType: 'model_inference' | 'transformation' | 'condition' |
        'parallel' | 'aggregation';
  config: object;
  dependsOn: string[];
  conditionExpression?: string;
}
```

---

11. ResponseSynthesisService (`response-synthesis.service.ts`)

Purpose: Synthesizes responses from multiple AI models.

Synthesis Strategies:

Strategy	Description	Best For
`best_of`	Select highest confidence response	Quality-critical
`vote`	Majority voting on answer	Factual questions

Strategy	Description	Best For
`weighted`	Confidence × (1/latency) weighted	Balanced
`merge`	AI combines all responses	Complex analysis

Merge Algorithm:

```
1. Collect all responses with metadata
2. Extract key points from each
3. Identify agreements and conflicts
4. Generate unified response
5. Apply conflict resolution
6. Calculate final confidence
```

Billing Services

12. BillingService (`billing.ts`)

Purpose: Manages credits, subscriptions, and billing.

Key Methods: - `getSubscription(tenantId: string): Promise<Subscription>` - `getCreditBalance(tenantId: string): Promise<CreditBalance>` - `addCredits(tenantId: string, amount: number, type: string): Promise<number>` - `useCredits(tenantId: string, amount: number): Promise<{success, newBalance}>` - `purchaseCredits(tenantId: string, amount: number, price: number): Promise<string>`

Subscription Tiers: | Tier | Monthly Price | Annual Price | Credits/User | |—— —| ——— ——|——— ——|——— ——| | Free | $0 | - | 100 | | Pro | $49 | $490 | 5,000 | | Team | $199 | $1,990 | 25,000 | | Enterprise | Custom | Custom | Custom |

Volume Discounts: | Credit Amount | Discount | Bonus Credits | |—— ——— —|——— —-| ——— ——— —| | 10-19 | 5% | 0 | | 20-49 | 10% | 0 | | 50-99 | 15% | 5% | | 100+ | 25% | 10% |

Transaction Types: - `purchase` - Credit purchase - `bonus` - Promotional credits - `refund` - Refunded credits - `usage` - Credits consumed - `transfer_in` / `transfer_out` - Credit transfers - `subscription_allocation` - Monthly allocation - `expiration` - Expired credits - `adjustment` - Manual adjustment

---

13. StorageBillingService (`storage-billing.ts`)

Purpose: Tracks storage costs per tenant.

Billable Storage: - Uploaded files - Generated artifacts - Session history - Conversation logs

Pricing: - $0.023 per GB/month (Standard) - $0.0125 per GB/month (Infrequent) - $0.004 per GB/month (Archive)

Cognitive Services

14. CognitiveBrainService (`cognitive-brain.service.ts`)

Purpose: High-level cognitive processing and reasoning.

Cognitive Capabilities: - Working memory management - Attention allocation - Abstract reasoning - Analogy formation - Concept learning

---

15. ReasoningEngine (`reasoning-engine.ts`)

Purpose: Chain-of-thought and multi-step reasoning.

Reasoning Modes: | Mode | Description | |— —|— — — —-| | `deductive` | From general to specific | | `inductive` | From specific to general | | `abductive` | Best explanation inference | | `analogical` | Similarity-based reasoning |

---

16. CausalReasoningService (`causal-reasoning.service.ts`)

Purpose: Causal inference and counterfactual reasoning.

Methods: - `identifyCauses(effect: string): Promise<Cause[]>` - `predictEffects(cause: string): Promise<Effect[]>` - `counterfactual(scenario: string, change: string): Promise<string>`

---

17. GoalPlanningService (`goal-planning.service.ts`)

Purpose: Goal decomposition and planning.

Planning Algorithm:

```
1. Parse high-level goal
2. Identify subgoals
3. Determine dependencies
4. Sequence actions
5. Allocate resources
6. Execute and monitor
```

---

18. MetacognitionService (`metacognition.service.ts`)

Purpose: Self-reflection and learning from mistakes.

Metacognitive Functions: - Confidence calibration - Error detection - Strategy selection - Performance monitoring

---

Memory Services

19. MemoryService (`memory-service.ts`)

Purpose: Persistent memory across sessions.

Memory Types: - **Short-term**: Current session context - **Long-term**: Cross-session knowledge - **Episodic**: Event-based memories - **Semantic**: Factual knowledge

---

20. EpisodicMemoryService (`episodic-memory.service.ts`)

Purpose: Event-based memory storage and retrieval.

Key Methods: - `recordEpisode(event: Episode): Promise<void>` - `retrieveRelevant(query: string, limit: number): Promise<Episode[]>` - `consolidate(): Promise<void>` - Memory optimization

---

21. MemoryConsolidationService (`memory-consolidation.service.ts`)

Purpose: Optimizes memory storage by consolidating similar memories.

---

22. TimeMachineService (`time-machine.ts`)

Purpose: Access historical state at any point in time.

Key Methods: - `getStateAt(timestamp: Date): Promise<SystemState>` - `getDiff(from: Date, to: Date): Promise<StateDiff>` - `restore(timestamp: Date): Promise<void>`

AGI Services

23. AGIOrchestratorService (`agi-orchestrator.service.ts`)

Purpose: Coordinates AGI capabilities across services.

---

24. AdvancedAGIService (`advanced-agi.service.ts`)

Purpose: Advanced AGI features including self-improvement.

---

25. AGICompleteService (`agi-complete.service.ts`)

Purpose: Complete AGI pipeline from input to output.

---

26. AGIExtensionsService (`agi-extensions.service.ts`)

Purpose: Extensible AGI capabilities.

Collaboration Services

27. CollaborationService (`collaboration.ts`)

Purpose: Real-time collaboration features.

WebSocket Events: | Event | Direction | Description | |—— —-|—— —— —–|—— —— —-| | `join_session` | Client→Server | Join collaborative session | | `leave_session` | Client→Server | Leave session | | `cursor_move` | Bidirectional | Cursor position update | | `content_update` | Bidirectional | Content change | | `user_joined` | Server→Client | New user notification | | `user_left` | Server→Client | User left notification |

---

28. ConcurrentSessionManager (`concurrent-session.ts`)

Purpose: Manages concurrent user sessions.

Key Methods: - `createSession(config: SessionConfig): Promise<string>` - `joinSession(sessionId: string, userId: string): Promise<void>` - `getSessionState(sessionId: string): Promise<SessionState>` - `broadcastUpdate(sessionId: string, update: Update): Promise<void>`

---

29. TeamService (`team-service.ts`)

Purpose: Team and organization management.

Key Methods: - `createTeam(tenantId: string, name: string): Promise<string>` - `addMember(teamId: string, userId: string, role: string): Promise<void>` - `getTeamMembers(teamId: string): Promise<Member[]>`

Additional Services (30-62)

#	Service	File	Purpose
30	NeuralEngine	`neural-engine.ts`	Neural network operations
31	AutoResolveService	`auto-resolve.ts`	Automatic conflict resolution
32	CanvasService	`canvas-service.ts`	Visual canvas artifacts
33	PersonaService	`persona-service.ts`	AI persona management

#	Service	File	Purpose
34	SchedulerService	`scheduler-service.ts`	Task scheduling
35	LicenseService	`license-service.ts`	License management
36	UnifiedModelRegistry	`unified-model-registry.ts`	Central model registry
37	GrandfatheringService	`grandfathering-service.ts`	Legacy migration
38	VoiceVideoService	`voice-video.ts`	Voice/video processing
39	ResultMergingService	`result-merging.ts`	Merge results
40	WorldModelService	`world-model.service.ts`	World state modeling
41	MultiAgentService	`multi-agent.service.ts`	Multi-agent coordination
42	TheoryOfMindService	`theory-of-mind.service.ts`	Mental state modeling
43	MultimodalBindingService	`multimodal-binding.service.ts`	Cross-modal binding
44	SkillExecutionService	`skill-execution.service.ts`	Skill execution
45	AutonomousAgentService	`autonomous-agent.service.ts`	Autonomous operations
46	ConsciousnessService	`consciousness.service.ts`	Consciousness modeling
47	ConfigEngineService	`config-engine.service.ts`	Configuration engine
48	SelfImprovementService	`self-improvement.service.ts`	Self-improvement
49	MoralCompassService	`moral-compass.service.ts`	Ethical reasoning
50	MLTrainingService	`ml-training.service.ts`	ML model training
51	LearningService	`learning.service.ts`	Learning data collection
52	FeedbackService	`feedback.service.ts`	User feedback
53	FeedbackLearningService	`feedback-learning.ts`	Learn from feedback
54	WorkflowProposalService	`workflow-proposals.ts`	Workflow improvements
55	AppIsolationService	`app-isolation.ts`	App-level isolation
56	LocalizationService	`localization.ts`	i18n support
57	MigrationApprovalService	`migration-approval.ts`	Migration approval

#	Service	File	Purpose
58	SuperiorOrchestrationService	`superior-orchestration.service.ts`	Superior responses
59	RadiantUnifiedService	`radiant-unified.service.ts`	Unified API
60	NeuralOrchestrationService	`neural-orchestration.ts`	Neural orchestration
61	AuditService	`audit.ts`	Audit logging
62	APIKeysService	`api-keys.ts`	API key management

RADIANT Database Schema Reference

Complete Migration Inventory (40 Migrations)

Migration Index

#	Migration	Tables Created	Purpose
001	initial_schema	tenants, users, administrators, invitations, approval_requests	Core platform tables
002	tenant_isolation	RLS policies	Row-level security
003	ai_models	ai_models, model_capabilities	Model registry
004	usage_billing	usage_records, invoices	Usage tracking
005	admin_approval	approval_workflows	Admin approvals
006	self_hosted_models	self_hosted_models, model_deployments	Self-hosted AI
007	external_providers	external_providers, provider_configs	Provider management
010	visual_ai_pipeline	visual_pipelines, pipeline_stages	Visual AI processing
011	brain_router	routing_rules, task_classifications	Request routing
012	metrics_analytics	metrics, analytics_snapshots	Analytics
013	neural_engine	neural_configs, neural_executions	Neural processing
014	error_logging	error_logs, error_patterns	Error tracking
015	credentials_registry	credentials, credential_rotations	Credential management

#	Migration	Tables Created	Purpose
016	think_tank	thinktank_sessions, thinktank_steps, thinktank_tools	Think Tank
017	concurrent_chat	chat_sessions, chat_messages	Chat management
018	realtime_collaboration	collaborations, collaboration_members	Real-time collab
019	persistent_memory	memories, memory_associations	Memory system
020	focus_personas	personas, persona_configs	AI personas
021	team_plans	teams, team_members, team_plans	Team management
022	provider_registry	providers, provider_health	Provider registry
023	time_machine	snapshots, snapshot_diffs	Time machine
024	orchestration_engine	workflow_definitions, workflow_tasks, workflow_executions, task_executions	Orchestration
025	license_management	licenses, license_activations	Licensing
026	unified_model_registry	unified_models, model_versions	Model registry
027	feedback_learning	feedback, learning_samples	Feedback system
028	neural_orchestration	neural_workflows, neural_steps	Neural orchestration
029	workflow_proposals	proposals, proposal_evidence	Workflow improvements
030	app_isolation	app_contexts, app_permissions	App isolation
031	internationalization	locales, translations	i18n
032	dynamic_configuration	configs, config_history	Dynamic config
033	billing_credits	credit_balances, credit_transactions, credit_purchases	Credits
034	storage_billing	storage_usage, storage_costs	Storage billing
035	versioned_subscriptions	subscription_tiers, subscriptions	Subscriptions
036	dual_admin_approval	dual_approvals, approval_chains	Dual approval
037	canvas_artifacts	canvases, canvas_elements	Canvas
038	scheduled_prompts	scheduled_prompts, prompt_executions	Scheduling
039	auto_resolve	auto_resolutions, resolution_rules	Auto-resolve

#	Migration	Tables Created	Purpose
040	model_selection_pricing	model_pricing, selection_history	Pricing
041	admin_billing_enhancements	billing_reports, revenue_tracking	Billing enhancements

Core Tables (Migration 001)

tenants

Primary multi-tenant organization table.

```
CREATE TABLE tenants (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) NOT NULL UNIQUE,
    display_name VARCHAR(200) NOT NULL,
    domain VARCHAR(255),
    settings JSONB NOT NULL DEFAULT '{}',
    status VARCHAR(20) NOT NULL DEFAULT 'active'
        CHECK (status IN ('active', 'suspended', 'pending')),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_tenants_name ON tenants(name);
CREATE INDEX idx_tenants_domain ON tenants(domain) WHERE domain IS NOT
        NULL;
CREATE INDEX idx_tenants_status ON tenants(status);
```

Settings JSON Structure:

```
{
  "branding": {
    "logo_url": "string",
    "primary_color": "#hex",
    "company_name": "string"
  },
  "limits": {
    "max_users": 100,
    "max_api_keys": 10,
    "monthly_token_limit": 1000000
  },
  "features": {
    "think_tank_enabled": true,
    "orchestration_enabled": true,
    "collaboration_enabled": true
  },
```

```
  "compliance": {
    "hipaa_mode": false,
    "data_retention_days": 90
  }
}
```

---

users

End users within tenants.

```
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    cognito_user_id VARCHAR(128) NOT NULL,
    email VARCHAR(255) NOT NULL,
    display_name VARCHAR(200),
    role VARCHAR(50) NOT NULL DEFAULT 'user'
        CHECK (role IN ('user', 'power_user', 'admin')),
    status VARCHAR(20) NOT NULL DEFAULT 'active'
        CHECK (status IN ('active', 'suspended', 'pending')),
    settings JSONB NOT NULL DEFAULT '{}',
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    UNIQUE (tenant_id, email),
    UNIQUE (cognito_user_id)
);

-- Indexes
CREATE INDEX idx_users_tenant_id ON users(tenant_id);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_cognito_user_id ON users(cognito_user_id);
CREATE INDEX idx_users_status ON users(status);
```

User Roles: | Role | Permissions | |— —|— — — —-| | `user` | Basic API access, own resources | | `power_user` | + Create API keys, advanced features | | `admin` | + Manage users, view analytics |

---

administrators

Platform administrators (separate from tenant users).

```
CREATE TABLE administrators (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    cognito_user_id VARCHAR(128) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE,
    display_name VARCHAR(200) NOT NULL,
    role VARCHAR(50) NOT NULL DEFAULT 'admin'
        CHECK (role IN ('super_admin', 'admin', 'operator',
        'auditor')),
```

```
    permissions TEXT[] NOT NULL DEFAULT '{}',
    mfa_enabled BOOLEAN NOT NULL DEFAULT false,
    last_login_at TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    invited_by UUID REFERENCES administrators(id)
);
```

Admin Roles: | Role | Description | Permissions | |— —|— — — —-|— — — —-| | `super_admin` | Full platform access | All operations | | `admin` | Standard admin | Manage tenants, users, models | | `operator` | Operations | View logs, manage deployments | | `auditor` | Read-only audit | View all, modify none |

---

approval_requests

Two-person approval system.

```
CREATE TABLE approval_requests (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    requester_id UUID NOT NULL REFERENCES administrators(id),
    action_type VARCHAR(100) NOT NULL,
    resource_type VARCHAR(100) NOT NULL,
    resource_id VARCHAR(255),
    payload JSONB NOT NULL DEFAULT '{}',
    status VARCHAR(20) NOT NULL DEFAULT 'pending'
        CHECK (status IN ('pending', 'approved', 'rejected',
        'expired')),
    required_approvals INTEGER NOT NULL DEFAULT 1,
    approvals JSONB NOT NULL DEFAULT '[]',
    expires_at TIMESTAMPTZ NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

Action Types Requiring Approval: - `delete_tenant` - Delete a tenant - `modify_billing` - Change billing settings - `grant_super_admin` - Elevate to super admin - `bulk_data_export` - Export all data - `disable_security_feature` - Disable security

---

Think Tank Tables (Migration 016)

thinktank_sessions

Problem-solving session tracking.

```
CREATE TABLE thinktank_sessions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
```

```
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    problem_summary TEXT,
    domain VARCHAR(50),
    complexity VARCHAR(20) CHECK (complexity IN ('low', 'medium',
        'high', 'extreme')),
    total_steps INTEGER DEFAULT 0,
    avg_confidence DECIMAL(3, 2),
    solution_found BOOLEAN DEFAULT false,
    total_tokens INTEGER DEFAULT 0,
    total_cost DECIMAL(10, 6) DEFAULT 0,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    completed_at TIMESTAMPTZ
);

-- Indexes
CREATE INDEX idx_thinktank_sessions_tenant ON
        thinktank_sessions(tenant_id, created_at DESC);
CREATE INDEX idx_thinktank_sessions_user ON
        thinktank_sessions(user_id);

-- RLS
ALTER TABLE thinktank_sessions ENABLE ROW LEVEL SECURITY;
CREATE POLICY thinktank_sessions_isolation ON thinktank_sessions
    FOR ALL USING (tenant_id =
        current_setting('app.current_tenant_id', true)::uuid);
```

Domains: - `research` - Academic research - `engineering` - Technical problems - `analytical` - Data analysis - `creative` - Creative tasks - `legal` - Legal analysis - `medical` - Medical queries (HIPAA) - `business` - Business strategy - `general` - General problems

---

thinktank_steps

Individual reasoning steps within sessions.

```
CREATE TABLE thinktank_steps (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    session_id UUID NOT NULL REFERENCES thinktank_sessions(id) ON
        DELETE CASCADE,
    step_number INTEGER NOT NULL,
    step_type VARCHAR(50) NOT NULL
        CHECK (step_type IN ('decompose', 'reason', 'execute',
        'verify', 'synthesize')),
    description TEXT,
    reasoning TEXT,
    result TEXT,
    confidence DECIMAL(3, 2) CHECK (confidence >= 0 AND confidence <=
        1),
    model_used VARCHAR(100),
    tokens_used INTEGER,
    duration_ms INTEGER,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
```

```
);

-- Indexes
CREATE INDEX idx_thinktank_steps_session ON
        thinktank_steps(session_id, step_number);

-- RLS
ALTER TABLE thinktank_steps ENABLE ROW LEVEL SECURITY;
CREATE POLICY thinktank_steps_isolation ON thinktank_steps
    FOR ALL USING (
        session_id IN (SELECT id FROM thinktank_sessions
                       WHERE tenant_id =
        current_setting('app.current_tenant_id', true)::uuid)
    );
```

Step Types: | Type | Description | Typical Model | |— —|— — — — -|— — — — — —|| `decompose` | Break problem into parts | Claude 3.5 | | `reason` | Chain-of-thought reasoning | o1, Claude | | `execute` | Execute solution step | Task-specific | | `verify` | Verify result accuracy | Different model | | `synthesize` | Combine into final answer | Claude 3.5 |

---

thinktank_tools

Available tools for Think Tank.

```
CREATE TABLE thinktank_tools (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    tool_name VARCHAR(100) NOT NULL UNIQUE,
    tool_type VARCHAR(50) NOT NULL,
    description TEXT,
    parameters_schema JSONB NOT NULL DEFAULT '{}',
    implementation TEXT,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Default tools
INSERT INTO thinktank_tools (tool_name, tool_type, description,
        parameters_schema) VALUES
    ('web_search', 'search', 'Search the web for information',
        '{"query": "string"}'),
    ('calculator', 'compute', 'Perform mathematical calculations',
        '{"expression": "string"}'),
    ('code_executor', 'compute', 'Execute code snippets',
        '{"language": "string", "code": "string"}'),
    ('file_reader', 'io', 'Read file contents', '{"path": "string"}'),
    ('api_caller', 'network', 'Make API requests', '{"url": "string",
        "method": "string", "body": "object"}');
```

Orchestration Tables (Migration 024)

workflow_definitions

Workflow pattern definitions.

```
CREATE TABLE workflow_definitions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workflow_id VARCHAR(100) NOT NULL UNIQUE,
    name VARCHAR(200) NOT NULL,
    description TEXT,
    category VARCHAR(50) NOT NULL
        CHECK (category IN ('generation', 'analysis',
        'transformation', 'pipeline', 'custom')),
    version VARCHAR(20) NOT NULL DEFAULT '1.0.0',

    dag_definition JSONB NOT NULL DEFAULT '{}',
    input_schema JSONB NOT NULL DEFAULT '{}',
    output_schema JSONB NOT NULL DEFAULT '{}',
    default_parameters JSONB NOT NULL DEFAULT '{}',

    timeout_seconds INTEGER DEFAULT 3600,
    max_retries INTEGER DEFAULT 3,
    min_tier INTEGER DEFAULT 1,

    is_active BOOLEAN DEFAULT true,
    requires_audit_trail BOOLEAN DEFAULT false,
    hipaa_compliant BOOLEAN DEFAULT false,

    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    created_by UUID
);
```

DAG Definition Structure:

```
{
  "nodes": [
    {
      "taskId": "decompose",
      "taskType": "model_inference",
      "modelId": "anthropic/claude-3-5-sonnet",
      "config": {
        "systemPrompt": "Break down the problem...",
        "temperature": 0.3
      },
      "dependsOn": []
    },
    {
      "taskId": "solve_part_1",
      "taskType": "model_inference",
```

```
      "dependsOn": ["decompose"]
    }
  ],
  "edges": [
    {"from": "decompose", "to": "solve_part_1"}
  ]
}
```

---

workflow_tasks

Individual tasks within workflows.

```
CREATE TABLE workflow_tasks (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workflow_id UUID NOT NULL REFERENCES workflow_definitions(id) ON
        DELETE CASCADE,
    task_id VARCHAR(100) NOT NULL,
    name VARCHAR(200) NOT NULL,
    description TEXT,

    task_type VARCHAR(50) NOT NULL
        CHECK (task_type IN ('model_inference', 'transformation',
        'condition',
                             'parallel', 'aggregation',
        'external_api', 'human_review')),
    model_id VARCHAR(100),
    service_id VARCHAR(100),

    config JSONB NOT NULL DEFAULT '{}',
    input_mapping JSONB DEFAULT '{}',
    output_mapping JSONB DEFAULT '{}',

    sequence_order INTEGER DEFAULT 0,
    depends_on TEXT[] DEFAULT '{}',
    condition_expression TEXT,
    timeout_seconds INTEGER DEFAULT 300,

    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    UNIQUE(workflow_id, task_id)
);
```

Task Types: | Type | Description | Example | |— —|— — — —-|— — —| | `model_inference` | AI model call | Generate text | | `transformation` | Data transformation | Format output | | `condition` | Conditional branching | If confidence > 0.8 | | `parallel` | Parallel execution | Call 3 models | | `aggregation` | Combine results | Merge responses | | `external_api` | External API call | Web search | | `human_review` | Human-in-the-loop | Approval step |

---

workflow_executions

Workflow execution tracking.

```
CREATE TABLE workflow_executions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workflow_id UUID NOT NULL REFERENCES workflow_definitions(id),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    user_id UUID NOT NULL,

    status VARCHAR(20) NOT NULL DEFAULT 'pending'
        CHECK (status IN ('pending', 'running', 'paused', 'completed',
        'failed', 'cancelled')),

    input_parameters JSONB NOT NULL DEFAULT '{}',
    resolved_parameters JSONB DEFAULT '{}',
    output_data JSONB,

    error_message TEXT,
    error_details JSONB,

    started_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,
    duration_ms INTEGER,

    estimated_cost_usd DECIMAL(10, 4),
    actual_cost_usd DECIMAL(10, 4),

    checkpoint_data JSONB,
    priority INTEGER DEFAULT 5 CHECK (priority >= 1 AND priority <=
        10),

    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- RLS
ALTER TABLE workflow_executions ENABLE ROW LEVEL SECURITY;
CREATE POLICY workflow_executions_isolation ON workflow_executions
    FOR ALL USING (tenant_id =
        current_setting('app.current_tenant_id', true)::uuid);
```

---

task_executions

Individual task execution tracking.

```
CREATE TABLE task_executions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workflow_execution_id UUID NOT NULL REFERENCES
        workflow_executions(id) ON DELETE CASCADE,
    task_id VARCHAR(100) NOT NULL,
```

```
    status VARCHAR(20) NOT NULL DEFAULT 'pending'
        CHECK (status IN ('pending', 'running', 'completed', 'failed',
        'skipped', 'retrying')),
    attempt_number INTEGER DEFAULT 1,

    input_data JSONB,
    output_data JSONB,

    error_message TEXT,
    error_code VARCHAR(50),

    started_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,
    duration_ms INTEGER,

    resource_usage JSONB DEFAULT '{}',
    cost_usd DECIMAL(10, 4),

    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

Billing Tables (Migrations 033-035)

credit_balances

Tenant credit balance tracking.

```
CREATE TABLE credit_balances (
    tenant_id UUID PRIMARY KEY REFERENCES tenants(id) ON DELETE
        CASCADE,
    balance DECIMAL(12, 4) NOT NULL DEFAULT 0,
    lifetime_purchased DECIMAL(12, 4) NOT NULL DEFAULT 0,
    lifetime_used DECIMAL(12, 4) NOT NULL DEFAULT 0,
    lifetime_bonus DECIMAL(12, 4) NOT NULL DEFAULT 0,
    low_balance_alert_threshold DECIMAL(12, 4),
    last_low_balance_alert TIMESTAMPTZ,
    auto_purchase_enabled BOOLEAN DEFAULT false,
    auto_purchase_threshold DECIMAL(12, 4),
    auto_purchase_amount DECIMAL(12, 4),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

---

credit_transactions

Credit transaction history.

```sql
CREATE TABLE credit_transactions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    transaction_type VARCHAR(30) NOT NULL
        CHECK (transaction_type IN ('purchase', 'bonus', 'refund',
        'usage',
                                    'transfer_in', 'transfer_out',
                                    'subscription_allocation',
        'expiration', 'adjustment')),
    amount DECIMAL(12, 4) NOT NULL,
    balance_after DECIMAL(12, 4) NOT NULL,
    description TEXT,
    reference_id VARCHAR(255),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_credit_transactions_tenant ON
        credit_transactions(tenant_id, created_at DESC);
```

---

subscription_tiers

Available subscription plans.

```sql
CREATE TABLE subscription_tiers (
    id VARCHAR(50) PRIMARY KEY,
    display_name VARCHAR(100) NOT NULL,
    description TEXT,
    price_monthly DECIMAL(10, 2),
    price_annual DECIMAL(10, 2),
    included_credits_per_user DECIMAL(10, 2) NOT NULL DEFAULT 0,
    features JSONB NOT NULL DEFAULT '{}',
    limits JSONB NOT NULL DEFAULT '{}',
    is_public BOOLEAN DEFAULT true,
    sort_order INTEGER DEFAULT 0,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Default tiers
INSERT INTO subscription_tiers (id, display_name, price_monthly,
        price_annual, included_credits_per_user, features) VALUES
('free', 'Free', 0, NULL, 100, '{"think_tank": false, "orchestration":
        false, "models": ["gpt-4o-mini", "claude-3-haiku"]}'),
('pro', 'Pro', 49, 490, 5000, '{"think_tank": true, "orchestration":
        true, "models": "all"}'),
('team', 'Team', 199, 1990, 25000, '{"think_tank": true,
        "orchestration": true, "collaboration": true, "models":
        "all"}'),
('enterprise', 'Enterprise', NULL, NULL, 0, '{"think_tank": true,
        "orchestration": true, "collaboration": true, "models": "all",
        "custom_models": true}');
```

---

subscriptions

Active tenant subscriptions.

```
CREATE TABLE subscriptions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    tier_id VARCHAR(50) NOT NULL REFERENCES subscription_tiers(id),
    status VARCHAR(20) NOT NULL DEFAULT 'active'
        CHECK (status IN ('active', 'cancelled', 'past_due',
        'trialing', 'paused')),
    billing_cycle VARCHAR(10) NOT NULL CHECK (billing_cycle IN
        ('monthly', 'annual')),
    seats_purchased INTEGER NOT NULL DEFAULT 1,
    seats_used INTEGER NOT NULL DEFAULT 0,
    current_period_start TIMESTAMPTZ NOT NULL,
    current_period_end TIMESTAMPTZ NOT NULL,
    cancel_at_period_end BOOLEAN DEFAULT false,
    cancelled_at TIMESTAMPTZ,
    stripe_customer_id VARCHAR(255),
    stripe_subscription_id VARCHAR(255),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

Row-Level Security (Migration 002)

RLS Pattern

All tenant-scoped tables use this pattern:

```
-- Enable RLS on table
ALTER TABLE {table_name} ENABLE ROW LEVEL SECURITY;

-- Create isolation policy
CREATE POLICY {table_name}_isolation ON {table_name}
    FOR ALL USING (tenant_id =
        current_setting('app.current_tenant_id', true)::uuid);
```

Setting Tenant Context

Every request sets the tenant context before queries:

```
SET app.current_tenant_id = '{tenant_uuid}';
```

Tables with RLS Enabled:

- users
- thinktank_sessions

- thinktank_steps
- workflow_executions
- task_executions
- credit_transactions
- subscriptions
- chat_sessions
- chat_messages
- collaborations
- memories
- feedback
- api_keys
- (all tenant-scoped tables)

Common Patterns

Updated At Trigger

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Apply to tables
CREATE TRIGGER update_{table}_updated_at
    BEFORE UPDATE ON {table}
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

Soft Delete Pattern

```
-- Add columns
deleted_at TIMESTAMPTZ,
deleted_by UUID,

-- Query with filter
WHERE deleted_at IS NULL
```

Audit Columns

```
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
created_by UUID,
updated_by UUID
```

Swift Deployer App Reference

App Architecture

Overview

The Swift Deployer is a native macOS application for deploying and managing RADIANT infrastructure to AWS.

Requirements: - macOS 13.0 (Ventura) or later - Swift 5.9+ - Xcode 15+

File Structure (36 Files)

```
apps/swift-deployer/
├── Package.swift
└── Sources/RadiantDeployer/
    ├── RadiantDeployerApp.swift          # App entry point
    ├── AppState.swift                    # Global state management
    │
    ├── Config/
    │   └── RadiantConfig.swift           # App configuration
    │
    ├── Models/ (6 files)
    │   ├── Configuration.swift           # Deployment configuration
    │   ├── Credentials.swift             # AWS credentials model
    │   ├── Deployment.swift              # Deployment state model
    │   ├── DomainConfiguration.swift     # Domain settings
    │   ├── InstallationParameters.swift  # Install params
    │   └── ManagedApp.swift              # Managed app model
    │
    ├── Services/ (21 files)
    │   ├── AIAssistantService.swift      # AI deployment assistant
    │   ├── AIRegistryService.swift       # AI model registry
    │   ├── APIService.swift              # API communication
    │   ├── AWSService.swift              # AWS SDK wrapper
    │   ├── AuditLogger.swift             # Audit logging
    │   ├── CDKService.swift              # CDK deployment
    │   ├── CredentialService.swift       # Credential management
    │   ├── DNSService.swift              # DNS configuration
    │   ├── DatabaseService.swift         # Local SQLite/SQLCipher
    │   ├── DeploymentLockService.swift   # Deployment locking
    │   ├── DeploymentService.swift       # Main deployment logic
    │   ├── GitHubPackageRegistry.swift   # Package downloads
    │   ├── HealthCheckService.swift      # Health monitoring
    │   ├── LocalStorageManager.swift     # Encrypted local storage
    │   ├── MultiRegionService.swift      # Multi-region deployment
    │   ├── OnePasswordService.swift      # 1Password integration
    │   ├── PackageService.swift          # Package management
```

```
│   ├── SeedDataService.swift          # Database seeding
│   ├── SnapshotService.swift          # State snapshots
│   ├── TimeoutService.swift           # Timeout handling
│   └── VoiceInputService.swift        # Voice commands
│
├── Views/ (8+ files)
│   ├── ABTestingView.swift            # A/B testing config
│   ├── ContentView.swift              # Main content
│   ├── DeploymentView.swift           # Deployment UI
│   ├── SettingsView.swift             # App settings
│   └── ... (other views)
│
└── Components/ (4 files)
    ├── MacOSComponents.swift          # Design tokens & components
    ├── AppCommands.swift              # Menu bar commands
    ├── DataTableComponents.swift      # Table components
    └── DetailViewComponents.swift     # Detail view patterns
```

Models

Configuration.swift

Deployment configuration model.

```swift
struct DeploymentConfiguration: Codable, Sendable {
    var appId: String
    var environment: Environment
    var tier: Int
    var region: AWSRegion
    var domain: String?
    var enabledStacks: Set<StackName>
    var customParameters: [String: String]
}

enum Environment: String, Codable, CaseIterable, Sendable {
    case development = "dev"
    case staging = "staging"
    case production = "prod"
}

enum StackName: String, Codable, CaseIterable, Sendable {
    case networking
    case foundation
    case data
    case storage
    case auth
    case ai
    case api
    case admin
```

```
    case batch
    case collaboration
    case monitoring
    case security
    case webhooks
    case scheduledTasks
    case multiRegion
}
```

Credentials.swift

AWS credentials model.

```
struct AWSCredentials: Codable, Sendable {
    let accessKeyId: String
    let secretAccessKey: String
    let sessionToken: String?
    let region: String
    let profile: String?
    let expiresAt: Date?

    var isExpired: Bool {
        guard let expiresAt else { return false }
        return Date() > expiresAt
    }
}

struct CredentialProfile: Codable, Identifiable, Sendable {
    let id: UUID
    var name: String
    var credentials: AWSCredentials
    var isDefault: Bool
    var lastUsed: Date?
}
```

Deployment.swift

Deployment state tracking.

```
struct Deployment: Codable, Identifiable, Sendable {
    let id: UUID
    var configuration: DeploymentConfiguration
    var status: DeploymentStatus
    var stackStatuses: [StackName: StackStatus]
    var startedAt: Date
    var completedAt: Date?
    var errorMessage: String?
    var outputs: [String: String]
}

enum DeploymentStatus: String, Codable, Sendable {
```

```
    case pending
    case preparing
    case deploying
    case verifying
    case completed
    case failed
    case rollingBack
    case cancelled
}

enum StackStatus: String, Codable, Sendable {
    case pending
    case creating
    case updating
    case complete
    case failed
    case rollbackInProgress
    case rollbackComplete
    case deleted
}
```

Services

CDKService.swift

AWS CDK deployment service.

```
actor CDKService {
    private let shell: ShellService
    private let logger: AuditLogger

    // Deploy a single stack
    func deployStack(
        _ stack: StackName,
        config: DeploymentConfiguration,
        credentials: AWSCredentials
    ) async throws -> StackOutput {
        let command = buildCDKCommand(
            action: "deploy",
            stack: stack,
            config: config
        )

        return try await shell.execute(
            command,
            environment: credentials.asEnvironment()
        )
    }
```

```
    // Deploy all stacks in dependency order
    func deployAllStacks(
        config: DeploymentConfiguration,
        credentials: AWSCredentials,
        progressHandler: @Sendable (StackName, StackStatus) -> Void
    ) async throws -> DeploymentResult {
        let orderedStacks = topologicalSort(config.enabledStacks)
        var outputs: [StackName: StackOutput] = [:]

        for stack in orderedStacks {
            progressHandler(stack, .creating)
            do {
                outputs[stack] = try await deployStack(stack, config:
    config, credentials: credentials)
                progressHandler(stack, .complete)
            } catch {
                progressHandler(stack, .failed)
                throw DeploymentError.stackFailed(stack, error)
            }
        }

        return DeploymentResult(outputs: outputs)
    }

    // Stack dependency order
    private func topologicalSort(_ stacks: Set<StackName>) ->
        [StackName] {
        // networking → foundation → data/storage/auth → ai →
        api/admin → rest
        let order: [StackName] = [
            .networking, .foundation, .data, .storage, .auth,
            .ai, .api, .admin, .batch, .collaboration,
            .monitoring, .security, .webhooks, .scheduledTasks,
        .multiRegion
        ]
        return order.filter { stacks.contains($0) }
    }
}
```

DeploymentService.swift

Main deployment orchestration.

```
@MainActor
class DeploymentService: ObservableObject {
    @Published var currentDeployment: Deployment?
    @Published var deploymentHistory: [Deployment] = []
    @Published var isDeploying = false

    private let cdkService: CDKService
    private let healthService: HealthCheckService
    private let auditLogger: AuditLogger
```

```
    private let localStorage: LocalStorageManager

    func startDeployment(config: DeploymentConfiguration) async throws
        {
        guard !isDeploying else {
            throw DeploymentError.alreadyInProgress
        }

        isDeploying = true
        let deployment = Deployment(
            id: UUID(),
            configuration: config,
            status: .preparing,
            stackStatuses: [:],
            startedAt: Date()
        )
        currentDeployment = deployment

        do {
            // 1. Validate configuration
            try await validateConfiguration(config)

            // 2. Acquire deployment lock
            try await acquireLock(config.appId)

            // 3. Deploy stacks
            currentDeployment?.status = .deploying
            let result = try await cdkService.deployAllStacks(
                config: config,
                credentials: try await getCredentials(),
                progressHandler: { [weak self] stack, status in
                    Task { @MainActor in
                        self?.currentDeployment?.stackStatuses[stack]
        = status
                    }
                }
            )

            // 4. Verify deployment
            currentDeployment?.status = .verifying
            try await healthService.verifyDeployment(result)

            // 5. Complete
            currentDeployment?.status = .completed
            currentDeployment?.completedAt = Date()
            currentDeployment?.outputs = result.flatOutputs

        } catch {
            currentDeployment?.status = .failed
            currentDeployment?.errorMessage =
        error.localizedDescription
            throw error
```

```
            } finally {
                isDeploying = false
                if let deployment = currentDeployment {
                    deploymentHistory.append(deployment)
                    try? await localStorage.saveDeployment(deployment)
                }
            }
        }

        func rollback(deploymentId: UUID) async throws {
            // Rollback implementation
        }
    }
```

AIAssistantService.swift

AI-powered deployment assistant.

```
    actor AIAssistantService {
        private let apiService: APIService

        struct AssistantResponse: Sendable {
            let message: String
            let suggestions: [Suggestion]
            let actions: [SuggestedAction]
        }

        enum SuggestedAction: Sendable {
            case deployStack(StackName)
            case checkHealth
            case viewLogs(String)
            case runDiagnostics
            case contactSupport
        }

        // Get deployment guidance
        func getDeploymentGuidance(
            for config: DeploymentConfiguration,
            currentStatus: DeploymentStatus?
        ) async throws -> AssistantResponse {
            let prompt = buildGuidancePrompt(config: config, status:
            currentStatus)
            let response = try await apiService.chat(
                messages: [.init(role: .user, content: prompt)],
                model: "anthropic/claude-3-haiku"
            )
            return parseAssistantResponse(response)
        }

        // Diagnose deployment error
        func diagnoseError(
```

```
            error: Error,
            stackName: StackName,
            logs: String
        ) async throws -> AssistantResponse {
            let prompt = """
            Analyze this AWS CDK deployment error and provide:
            1. Root cause analysis
            2. Specific fix steps
            3. Prevention recommendations

            Stack: \(stackName.rawValue)
            Error: \(error.localizedDescription)
            Logs:
            \(logs.prefix(2000))
            """

            let response = try await apiService.chat(
                messages: [.init(role: .user, content: prompt)],
                model: "anthropic/claude-3-5-sonnet"
            )
            return parseAssistantResponse(response)
        }
    }
```

LocalStorageManager.swift

Encrypted local storage using SQLCipher.

```
actor LocalStorageManager {
    private let db: Connection
    private let encryptionKey: String

    init() throws {
        let path = LocalStorageManager.databasePath
        db = try Connection(path)
        encryptionKey = try KeychainService.getOrCreateDatabaseKey()
        try db.key(encryptionKey)
        try createTablesIfNeeded()
    }

    // Tables
    private func createTablesIfNeeded() throws {
        try db.execute("""
            CREATE TABLE IF NOT EXISTS deployments (
                id TEXT PRIMARY KEY,
                data BLOB NOT NULL,
                created_at INTEGER NOT NULL
            );

            CREATE TABLE IF NOT EXISTS credentials (
                id TEXT PRIMARY KEY,
```

```
                    profile_name TEXT NOT NULL,
                    encrypted_data BLOB NOT NULL,
                    is_default INTEGER DEFAULT 0,
                    last_used INTEGER
                );

                CREATE TABLE IF NOT EXISTS settings (
                    key TEXT PRIMARY KEY,
                    value TEXT NOT NULL
                );
            """)
        }

        // Save deployment
        func saveDeployment(_ deployment: Deployment) throws {
            let data = try JSONEncoder().encode(deployment)
            try db.run("""
                INSERT OR REPLACE INTO deployments (id, data, created_at)
                VALUES (?, ?, ?)
            """, deployment.id.uuidString, data,
            Date().timeIntervalSince1970)
        }

        // Get deployment history
        func getDeploymentHistory() throws -> [Deployment] {
            let rows = try db.prepare("""
                SELECT data FROM deployments ORDER BY created_at DESC
            LIMIT 100
            """)
            return try rows.compactMap { row in
                guard let data = row[0] as? Data else { return nil }
                return try JSONDecoder().decode(Deployment.self, from:
            data)
            }
        }

        // Credential management
        func saveCredentials(_ credentials: CredentialProfile) throws {
            let encrypted = try encrypt(credentials)
            try db.run("""
                INSERT OR REPLACE INTO credentials (id, profile_name,
            encrypted_data, is_default, last_used)
                VALUES (?, ?, ?, ?, ?)
            """, credentials.id.uuidString, credentials.name, encrypted,
                 credentials.isDefault ? 1 : 0,
            credentials.lastUsed?.timeIntervalSince1970)
        }
    }
```

HealthCheckService.swift

Deployment health verification.

```
actor HealthCheckService {
    struct HealthCheckResult: Sendable {
        let service: String
        let status: HealthStatus
        let latencyMs: Int?
        let message: String?
    }

    enum HealthStatus: Sendable {
        case healthy
        case degraded
        case unhealthy
        case unknown
    }

    func verifyDeployment(_ result: DeploymentResult) async throws {
        var checks: [HealthCheckResult] = []

        // Check API Gateway
        if let apiUrl = result.outputs["ApiUrl"] {
            checks.append(await checkEndpoint(apiUrl + "/health",
        service: "API Gateway"))
        }

        // Check LiteLLM
        if let litellmUrl = result.outputs["LiteLLMUrl"] {
            checks.append(await checkEndpoint(litellmUrl + "/health",
        service: "LiteLLM"))
        }

        // Check Database
        checks.append(await
        checkDatabase(result.outputs["DatabaseEndpoint"]))

        // Evaluate results
        let unhealthy = checks.filter { $0.status == .unhealthy }
        if !unhealthy.isEmpty {
            throw HealthCheckError.servicesUnhealthy(unhealthy)
        }
    }

    private func checkEndpoint(_ url: String, service: String) async -
        > HealthCheckResult {
        let start = Date()
        do {
            let (_, response) = try await URLSession.shared.data(from:
        URL(string: url)!)
            let httpResponse = response as! HTTPURLResponse
            let latency = Int(Date().timeIntervalSince(start) * 1000)

            return HealthCheckResult(
                service: service,
```

```
                status: httpResponse.statusCode == 200 ? .healthy :
        .degraded,
                latencyMs: latency,
                message: nil
            )
        } catch {
            return HealthCheckResult(
                service: service,
                status: .unhealthy,
                latencyMs: nil,
                message: error.localizedDescription
            )
        }
    }
}
```

Components

MacOSComponents.swift

Design tokens and reusable components.

```
// Design Tokens
enum RadiantSpacing {
    static let xxs: CGFloat = 2
    static let xs: CGFloat = 4
    static let sm: CGFloat = 8
    static let md: CGFloat = 12
    static let lg: CGFloat = 16
    static let xl: CGFloat = 24
}

enum RadiantRadius {
    static let sm: CGFloat = 4
    static let md: CGFloat = 8
    static let lg: CGFloat = 12
    static let xl: CGFloat = 16
}

// Status Badge Component
struct StatusBadge: View {
    let status: String
    let color: Color

    var body: some View {
        Text(status)
            .font(.caption)
            .fontWeight(.medium)
            .padding(.horizontal, RadiantSpacing.sm)
```

```
                .padding(.vertical, RadiantSpacing.xxs)
                .background(color.opacity(0.15))
                .foregroundColor(color)
                .cornerRadius(RadiantRadius.sm)
        }
    }

    // Progress Indicator
    struct DeploymentProgressView: View {
        let stacks: [StackName]
        let statuses: [StackName: StackStatus]

        var body: some View {
            VStack(alignment: .leading, spacing: RadiantSpacing.sm) {
                ForEach(stacks, id: \.self) { stack in
                    HStack {
                        statusIcon(for: statuses[stack] ?? .pending)
                        Text(stack.rawValue)
                            .font(.system(.body, design: .monospaced))
                        Spacer()
                        StatusBadge(
                            status: (statuses[stack] ??
    .pending).rawValue,
                            color: statusColor(statuses[stack] ??
    .pending)
                        )
                    }
                }
            }
        }
    }
```

AppCommands.swift

Menu bar commands with keyboard shortcuts.

```
    struct AppCommands: Commands {
        @ObservedObject var appState: AppState

        var body: some Commands {
            CommandGroup(replacing: .newItem) {
                Button("New Deployment") {
                    appState.showNewDeploymentSheet = true
                }
                .keyboardShortcut("n", modifiers: .command)

                Button("Import Configuration...") {
                    appState.importConfiguration()
                }
                .keyboardShortcut("i", modifiers: [.command, .shift])
            }
```

```
        CommandMenu("Deployment") {
            Button("Deploy All Stacks") {
                Task { await appState.deployAll() }
            }
            .keyboardShortcut("d", modifiers: [.command, .shift])
            .disabled(appState.isDeploying)

            Button("Stop Deployment") {
                appState.stopDeployment()
            }
            .keyboardShortcut(".", modifiers: .command)
            .disabled(!appState.isDeploying)

            Divider()

            Button("View Logs") {
                appState.showLogs = true
            }
            .keyboardShortcut("l", modifiers: [.command, .option])

            Button("Run Health Check") {
                Task { await appState.runHealthCheck() }
            }
            .keyboardShortcut("h", modifiers: [.command, .shift])
        }

        CommandMenu("AWS") {
            Button("Switch Profile...") {
                appState.showProfileSwitcher = true
            }
            .keyboardShortcut("p", modifiers: [.command, .option])

            Button("Refresh Credentials") {
                Task { await appState.refreshCredentials() }
            }
            .keyboardShortcut("r", modifiers: [.command, .shift])
        }
    }
}
```

UI Patterns (10 macOS Patterns)

The Swift Deployer follows these macOS design patterns:

1. **NavigationSplitView** - Sidebar + Content + Inspector
2. **Liquid Glass** - On navigation/controls only
3. **Toolbar-as-Command-Center** - Grouped actions + overflow
4. **Scroll Edge Effects** - Floating UI legibility

5. **Master List → Detail** - 3-level navigation
6. **Search as First-Class** - Toolbar trailing position
7. **Tables for Data** - Lists for collections
8. **Multi-Select + Context Menus** - Drag & drop support
9. **Full Menu Bar** - Keyboard shortcuts
10. **Settings Window + Inspectors** - macOS-native patterns # Admin Dashboard Reference

Dashboard Architecture

Technology Stack

- **Framework:** Next.js 14 (App Router)
- **Language:** TypeScript
- **Styling:** Tailwind CSS
- **Components:** shadcn/ui
- **Icons:** Lucide React
- **State:** React Query, Zustand
- **Forms:** React Hook Form, Zod

Page Structure

```
apps/admin-dashboard/
├── app/
│   ├── layout.tsx              # Root layout
│   ├── page.tsx                # Landing/login
│   └── (dashboard)/            # Dashboard routes (43 pages)
│       ├── layout.tsx          # Dashboard layout with sidebar
│       ├── page.tsx            # Overview dashboard
│       └── [module]/page.tsx   # Individual modules
├── components/
│   ├── ui/                     # shadcn/ui components
│   ├── workflow-editor/        # Visual workflow editor
│   └── shared/                 # Shared components
└── lib/
    ├── api.ts                  # API client
    ├── auth.ts                 # Auth utilities
    └── utils.ts                # Helper functions
```

Complete Page Inventory (43 Pages)

Core Administration

Page	Route	Purpose
Overview	`/`	System health, key metrics, quick actions

Page	Route	Purpose
Administrators	`/administrators`	Manage admin users, roles, permissions
Audit Logs	`/audit-logs`	View all system audit events
AWS Logs	`/aws-logs`	CloudWatch log viewer
Security	`/security`	Security settings, WAF, compliance
Settings	`/settings`	Platform configuration
System Config	`/system-config`	Advanced system settings

AI & Models

Page	Route	Purpose
Models	`/models`	AI model configuration
Model Metadata	`/model-metadata`	Model capabilities & pricing
User Models	`/user-models`	Per-tenant model access
Providers	`/providers`	AI provider management

Orchestration

Page	Route	Purpose
Orchestration	`/orchestration`	Workflow management
Orchestration Patterns	`/orchestration-patterns`	49 patterns library
Orchestration Editor	`/orchestration-patterns/editor`	Visual workflow editor

Think Tank

Page	Route	Purpose
Think Tank	`/thinktank`	Session management
Cognition	`/cognition`	Cognitive settings
Cognitive Brain	`/cognitive-brain`	Brain configuration
Consciousness	`/consciousness`	Consciousness monitoring
Metacognition	`/metacognition`	Self-reflection settings
Planning	`/planning`	Goal planning
World Model	`/world-model`	World model state

Billing & Cost

Page	Route	Purpose
Billing	`/billing`	Revenue, invoices, subscriptions
Cost	`/cost`	Cost analytics, budgets

Analytics & Monitoring

Page	Route	Purpose
Analytics	`/analytics`	Usage analytics
Reports	`/reports`	Generated reports
Health	`/health`	System health dashboard
Deployments	`/deployments`	Deployment history

AGI & Learning

Page	Route	Purpose
Agents	`/agents`	Autonomous agents
Learning	`/learning`	ML training data
ML Training	`/ml-training`	Model training jobs
Self-Improvement	`/self-improvement`	Self-improvement logs
Moral Compass	`/moral-compass`	Ethical guidelines
Feedback	`/feedback`	User feedback

Collaboration & Features

Page	Route	Purpose
Time Machine	`/time-machine`	Historical state access
Storage	`/storage`	File storage management
Notifications	`/notifications`	Notification settings
Localization	`/localization`	i18n management
Configuration	`/configuration`	Dynamic configuration
Compliance	`/compliance`	Compliance dashboard
Geographic	`/geographic`	Geographic settings
Multi-Region	`/multi-region`	Multi-region config
Experiments	`/experiments`	A/B testing
Migrations	`/migrations`	Database migrations
Services	`/services`	Service status
Request Handler	`/request-handler`	Request routing

Key Pages Detail

Overview Dashboard (`/`)

Metrics Displayed: - Active tenants (24h) - Total API requests (24h) - Total tokens processed - Revenue (MTD) - Error rate - Average latency

Quick Actions: - View recent errors - Check provider health - Review pending approvals - Generate report

Charts: - Requests over time (7d) - Token usage by model - Revenue trend - Error rate trend

---

Models Page (`/models`)

Features: - List all 106+ models - Filter by provider, capability - Enable/disable models - Set model pricing overrides - Configure fallback chains - View usage statistics

Model Card Display:

```
┌─────────────────────────────────────────────┐
│ anthropic/claude-3-5-sonnet                 │
│ ─────────────────────────────────────────── │
│ Provider: Bedrock (primary), LiteLLM (fb)   │
│ Capabilities: reasoning, coding, vision     │
│ Context: 200K tokens                        │
│ Pricing: $3.00/$15.00 per 1M tokens         │
│ Status: ● Enabled                           │
│ Usage (24h): 1.2M tokens                    │
│                                             │
│ [Configure] [Disable] [View Stats]          │
└─────────────────────────────────────────────┘
```

Orchestration Patterns Page (`/orchestration-patterns`)

Features: - Browse 49 patterns by category - Search patterns - View pattern details - Edit pattern workflows - Create custom patterns - View execution statistics

Pattern Categories Tabs: - Consensus & Aggregation (7) - Debate & Deliberation (7) - Critique & Refinement (7) - Verification & Validation (7) - Decomposition (7) - Specialized Reasoning (7) - Multi-Model Routing (4) - Ensemble Methods (3)

Pattern Detail View:

```
┌─────────────────────────────────────────────────────────────┐
│ AI Debate                                  [Edit] [Test]    │
│ ─────────────────────────────────────────────────────────── │
│ Category: Debate & Deliberation                             │
│ Quality Improvement: +25-40%                                │
```

```
| Typical Latency: High (10-30s)                                  |
| Min Models: 3                                                   |
|                                                                 |
| Description:                                                    |
| Two AI models debate opposing positions while a third           |
| model judges the arguments and synthesizes a final answer.      |
|                                                                 |
| Best For:                                                       |
| • Controversial topics                                          |
| • Complex decisions                                             |
| • Exploring multiple perspectives                               |
|                                                                 |
| Workflow Steps:                                                 |
| 1. Generate Pro Argument (Claude)                               |
| 2. Generate Con Argument (GPT-4o)                               |
| 3. Judge Arguments (Claude - thinking mode)                     |
| 4. Synthesize Final Answer                                      |
|                                                                 |
| Executions (30d): 1,247  |  Avg Quality: 0.89                   |
└─────────────────────────────────────────────────────────────────┘
```

Visual Workflow Editor (`/orchestration-patterns/editor`)

Features: - Drag-and-drop workflow design - 16 method palette - Node connection editing - Step configuration (4 tabs) - Zoom, pan, fit controls - Test execution - Save/load workflows

Method Palette (16 Methods): | Method | Category | Description | |— —–|— — —-| — — — —-| | Generate | Core | Generate text response | | Analyze | Core | Analyze input | | Transform | Core | Transform data | | Validate | Core | Validate output | | Critique | Refinement | Critique response | | Refine | Refinement | Improve response | | Decompose | Decomposition | Break into parts | | Synthesize | Aggregation | Combine results | | Judge | Evaluation | Evaluate quality | | Vote | Consensus | Majority voting | | Debate_Pro | Debate | Pro argument | | Debate_Con | Debate | Con argument | | Verify | Verification | Fact-check | | Search | External | Web search | | Execute_Code | External | Run code | | Custom | Custom | Custom logic |

Step Configuration Tabs: 1. **General** - Name, order, model, output variable 2. **Parameters** - Method-specific parameters 3. **Advanced** - Conditions, iterations, dependencies 4. **Parallel** - Parallel execution settings, AGI selection

---

Billing Page (`/billing`)

Sections:

Revenue Overview: - Monthly Recurring Revenue (MRR) - Annual Recurring Revenue (ARR) - Revenue growth % - Churn rate

Subscription Management: - Active subscriptions by tier - Upcoming renewals - Cancelled subscriptions - Trial conversions

Credit Management: - Total credits sold - Credits consumed - Credit purchase history - Low balance alerts

Invoice Management: - Generate invoices - View invoice history - Export to CSV - Send invoice reminders

---

Analytics Page (`/analytics`)

Dashboard Sections:

Usage Analytics: - Requests by model - Tokens by tenant - Peak usage times - Geographic distribution

Performance Analytics: - Latency percentiles (p50, p95, p99) - Error rates by endpoint - Provider availability - Cache hit rates

Business Analytics: - Cost per request - Revenue per tenant - Feature adoption - User engagement

Custom Reports: - Date range selection - Dimension grouping - Metric selection - Export options (CSV, PDF, JSON)

Security Page (`/security`)

Sections:

Authentication: - Cognito configuration - MFA enforcement - Session settings - Password policies

API Security: - Rate limiting rules - IP allowlists - API key management - Request validation

Compliance: - SOC2 status - HIPAA mode toggle - Data retention settings - Audit log retention

WAF Configuration: - Rule management - Blocked requests - Rate limit thresholds - Custom rules

---

Think Tank Page (`/thinktank`)

Features: - View all sessions across tenants - Filter by domain, status, confidence - Session detail view - Step-by-step reasoning display - Cost and token tracking

Session List View:

```
┌──────────────────────────────────────────────────────────────┐
│ Session ID    │ Tenant  │ Domain    │ Steps │ Conf  │ Cost  │
│ ──────────────│ ────────│ ──────────│ ──────│ ──────│ ──────│
│ abc123...     │ Acme    │ Engineering│ 6     │ 0.92  │ $0.45 │
│ def456...     │ Beta    │ Research  │ 8     │ 0.87  │ $0.72 │
```

```
| ghi789...     | Acme    | Legal     | 4     | 0.95  | $0.28 |
└──────────────────────────────────────────────────────────────┘
```

Session Detail View:

```
Problem: "Design a microservices architecture for 10M daily users"
Domain: Engineering | Complexity: High | Status: Completed

Step 1: Decompose [Claude 3.5] ✓ (conf: 0.94)
├─ Identified 5 sub-problems
└─ Duration: 2.3s | Tokens: 1,247

Step 2: Requirements Analysis [Claude + GPT-4o parallel] ✓ (conf:
0.91)
├─ Synthesized from 2 models
└─ Duration: 4.1s | Tokens: 3,892

Step 3-5: [...]

Step 6: Synthesize Final Solution [Claude 3.5 thinking] ✓ (conf: 0.89)
├─ Generated comprehensive solution
└─ Duration: 5.7s | Tokens: 2,156

Total: 6 steps | 18.2s | 12,453 tokens | $0.45
Final Confidence: 0.89
```

Component Library

Shared Components

DataTable: - Sortable columns - Pagination - Row selection - Export functionality - Column visibility toggle

StatusBadge: - Status indicator with color - Configurable variants - Icon support

MetricCard: - Large number display - Trend indicator - Comparison to previous period

Chart Components: - LineChart (time series) - BarChart (comparisons) - PieChart (distributions) - AreaChart (cumulative)

Form Components: - Input with validation - Select with search - DateRangePicker - JSONEditor - CodeEditor

---

API Integration

API Client (`lib/api.ts`)

```
import { QueryClient } from '@tanstack/react-query';

const API_BASE = process.env.NEXT_PUBLIC_API_URL;

export const apiClient = {
  // GET request
  async get<T>(path: string): Promise<T> {
    const response = await fetch(`${API_BASE}${path}`, {
      headers: await getAuthHeaders(),
    });
    if (!response.ok) throw new APIError(response);
    return response.json();
  },

  // POST request
  async post<T>(path: string, data: unknown): Promise<T> {
    const response = await fetch(`${API_BASE}${path}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        ...(await getAuthHeaders()),
      },
      body: JSON.stringify(data),
    });
    if (!response.ok) throw new APIError(response);
    return response.json();
  },

  // React Query hooks
  useModels: () => useQuery(['models'], () =>
        apiClient.get('/admin/models')),
  useTenants: () => useQuery(['tenants'], () =>
        apiClient.get('/admin/tenants')),
  useAnalytics: (range: string) =>
    useQuery(['analytics', range], () =>
        apiClient.get(`/admin/analytics?range=${range}`)),
};
```

Authentication (`lib/auth.ts`)

```
import { Amplify, Auth } from 'aws-amplify';

export async function getAuthHeaders(): Promise<Headers> {
  const session = await Auth.currentSession();
  return {
    'Authorization': `Bearer ${session.getIdToken().getJwtToken()}`,
```

```
  };
}

export function useAuth() {
  const [user, setUser] = useState<CognitoUser | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    Auth.currentAuthenticatedUser()
      .then(setUser)
      .catch(() => setUser(null))
      .finally(() => setLoading(false));
  }, []);

  return { user, loading, signIn, signOut };
}
```