Contents

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

Model ID	Provider	Capabilities	Cost ($/1K tokens)
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
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
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:

```
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
`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 Trial | $0 | - | 100 | | Individual | $19 | $190 | 1,000 | | 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
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
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