Contents

SECTION 34: DATABASE-DRIVEN ORCHESTRATION ENGINE (v4.1.0) 1

1

34.1 DATABASE SCHEMA 1
packages/database/migrations/034_orchestration_engine.sql 1
34.2 ALPHAFOLD 2 SEED DATA 8
packages/database/migrations/034a_seed_alphafold2.sql 8
34.3 ORCHESTRATION ENGINE SERVICE 10
packages/services/orchestration/OrchestrationEngine.ts 10

15

SECTION 34: DATABASE-DRIVEN ORCHESTRATION ENGINE (v4.1.0)

> **CRITICAL: This section replaces ALL hardcoded model configurations with database-driven management. All model configs, workflows, and orchestration parameters are stored in PostgreSQL. Administrators can add/edit/delete models entirely through the Admin Dashboard UI.**

34.1 DATABASE SCHEMA

packages/database/migrations/034_orchestration_engine.sql

```
-- ============================================================================
-- RADIANT v4.1.0 - DATABASE-DRIVEN ORCHESTRATION ENGINE
-- ============================================================================
-- This migration creates the foundation for dynamic model management.
-- ALL model configurations are stored in PostgreSQL - NO HARDCODING.
-- ============================================================================

-- AI Model Registry (Replaces hardcoded TypeScript configs)
CREATE TABLE IF NOT EXISTS ai_models (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  model_id VARCHAR(100) UNIQUE NOT NULL,
  name VARCHAR(100) NOT NULL,
  display_name VARCHAR(200) NOT NULL,
  description TEXT,
  category VARCHAR(50) NOT NULL,
  specialty VARCHAR(50),
  provider_type VARCHAR(20) NOT NULL DEFAULT 'self_hosted',

  deployment_config JSONB NOT NULL DEFAULT '{}',
```

```
  parameters BIGINT DEFAULT 0,
  accuracy VARCHAR(200),
  benchmark TEXT,
  capabilities TEXT[] DEFAULT '{}',
  input_formats TEXT[] DEFAULT '{}',
  output_formats TEXT[] DEFAULT '{}',
  architecture JSONB DEFAULT '{}',
  performance_metrics JSONB DEFAULT '{}',

  thermal_config JSONB NOT NULL DEFAULT '{"defaultState":"OFF","scaleToZeroAfterMinutes":15,"wa
  current_thermal_state VARCHAR(20) DEFAULT 'OFF',
  last_thermal_change TIMESTAMPTZ,

  pricing_config JSONB NOT NULL DEFAULT '{"hourlyRate":0,"perRequest":0,"markup":0.75}',

  min_tier INTEGER DEFAULT 1,
  requires_gpu BOOLEAN DEFAULT false,
  gpu_memory_gb INTEGER DEFAULT 0,
  enabled BOOLEAN DEFAULT true,
  status VARCHAR(20) DEFAULT 'active',
  version VARCHAR(50),
  repository VARCHAR(500),
  release_date DATE,

  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW(),
  created_by UUID REFERENCES administrators(id),
  updated_by UUID REFERENCES administrators(id)
);

-- License Tracking
CREATE TABLE IF NOT EXISTS model_licenses (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  model_id UUID NOT NULL REFERENCES ai_models(id) ON DELETE CASCADE,
  license_type VARCHAR(50) NOT NULL,
  license_spdx VARCHAR(50) NOT NULL,
  license_url VARCHAR(500),
  commercial_use BOOLEAN DEFAULT true,
  commercial_notes TEXT,
  attribution_required BOOLEAN DEFAULT false,
  attribution_text TEXT,
  share_alike BOOLEAN DEFAULT false,
  compliance_status VARCHAR(20) DEFAULT 'compliant',
  last_compliance_review TIMESTAMPTZ,
  reviewed_by UUID REFERENCES administrators(id),
  compliance_notes TEXT,
  expires_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW(),
```

```
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Model Dependencies
CREATE TABLE IF NOT EXISTS model_dependencies (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  model_id UUID NOT NULL REFERENCES ai_models(id) ON DELETE CASCADE,
  dependency_type VARCHAR(50) NOT NULL,
  dependency_name VARCHAR(200) NOT NULL,
  dependency_size_gb DECIMAL(10,2),
  dependency_license VARCHAR(50),
  required BOOLEAN DEFAULT true,
  notes TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Workflow Definitions
CREATE TABLE IF NOT EXISTS workflow_definitions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  workflow_id VARCHAR(100) UNIQUE NOT NULL,
  name VARCHAR(200) NOT NULL,
  description TEXT,
  category VARCHAR(50) NOT NULL,
  version VARCHAR(20) NOT NULL DEFAULT '1.0.0',
  dag_definition JSONB NOT NULL,
  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,
  enabled BOOLEAN DEFAULT true,
  requires_audit_trail BOOLEAN DEFAULT false,
  hipaa_compliant BOOLEAN DEFAULT false,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW(),
  created_by UUID REFERENCES administrators(id)
);

-- Workflow Tasks
CREATE TABLE IF NOT EXISTS workflow_tasks (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  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,
  model_id UUID REFERENCES ai_models(id),
```

```
  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 DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(workflow_id, task_id)
);

-- Workflow Parameters
CREATE TABLE IF NOT EXISTS workflow_parameters (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  workflow_id UUID NOT NULL REFERENCES workflow_definitions(id) ON DELETE CASCADE,
  parameter_name VARCHAR(100) NOT NULL,
  display_name VARCHAR(200),
  description TEXT,
  data_type VARCHAR(50) NOT NULL,
  default_value JSONB,
  validation_rules JSONB DEFAULT '{}',
  ui_component VARCHAR(50) DEFAULT 'text',
  ui_config JSONB DEFAULT '{}',
  user_configurable BOOLEAN DEFAULT true,
  admin_only BOOLEAN DEFAULT false,
  sequence_order INTEGER DEFAULT 0,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(workflow_id, parameter_name)
);

-- Workflow Executions
CREATE TABLE IF NOT EXISTS workflow_executions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  workflow_id UUID NOT NULL REFERENCES workflow_definitions(id),
  tenant_id UUID NOT NULL REFERENCES tenants(id),
  user_id UUID NOT NULL REFERENCES users(id),
  status VARCHAR(20) NOT NULL DEFAULT 'pending',
  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,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Task Executions
CREATE TABLE IF NOT EXISTS task_executions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  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',
  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),
  model_id UUID REFERENCES ai_models(id),
  model_endpoint VARCHAR(500),
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Orchestration Audit Log
CREATE TABLE IF NOT EXISTS orchestration_audit_log (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  entity_type VARCHAR(50) NOT NULL,
  entity_id UUID NOT NULL,
  action VARCHAR(50) NOT NULL,
  actor_id UUID REFERENCES administrators(id),
  actor_type VARCHAR(20) DEFAULT 'admin',
  previous_state JSONB,
  new_state JSONB,
  change_summary TEXT,
  ip_address INET,
  user_agent TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Indexes
CREATE INDEX IF NOT EXISTS idx_ai_models_category ON ai_models(category);
CREATE INDEX IF NOT EXISTS idx_ai_models_status ON ai_models(status, enabled);
```

```sql
CREATE INDEX IF NOT EXISTS idx_model_licenses_model ON model_licenses(model_id);
CREATE INDEX IF NOT EXISTS idx_model_licenses_compliance ON model_licenses(compliance_status);
CREATE INDEX IF NOT EXISTS idx_workflow_executions_tenant ON workflow_executions(tenant_id);
CREATE INDEX IF NOT EXISTS idx_workflow_executions_status ON workflow_executions(status);
CREATE INDEX IF NOT EXISTS idx_orchestration_audit_time ON orchestration_audit_log(created_at

-- Function: Get full model config (replaces hardcoded lookups)
CREATE OR REPLACE FUNCTION get_model_config(p_model_id VARCHAR)
RETURNS JSONB AS $$
DECLARE
  result JSONB;
BEGIN
  SELECT jsonb_build_object(
    'id', m.model_id,
    'name', m.name,
    'displayName', m.display_name,
    'description', m.description,
    'category', m.category,
    'providerType', m.provider_type,
    'deployment', m.deployment_config,
    'thermal', m.thermal_config,
    'pricing', m.pricing_config,
    'parameters', m.parameters,
    'accuracy', m.accuracy,
    'minTier', m.min_tier,
    'requiresGpu', m.requires_gpu,
    'status', m.status,
    'version', m.version,
    'currentThermalState', m.current_thermal_state,
    'licenses', (
      SELECT COALESCE(jsonb_agg(jsonb_build_object(
        'id', l.id, 'type', l.license_type, 'spdx', l.license_spdx,
        'commercialUse', l.commercial_use, 'complianceStatus', l.compliance_status
      )), '[]'::jsonb)
      FROM model_licenses l WHERE l.model_id = m.id
    ),
    'dependencies', (
      SELECT COALESCE(jsonb_agg(jsonb_build_object(
        'name', d.dependency_name, 'type', d.dependency_type,
        'sizeGb', d.dependency_size_gb, 'required', d.required
      )), '[]'::jsonb)
      FROM model_dependencies d WHERE d.model_id = m.id
    )
  ) INTO result
  FROM ai_models m
  WHERE m.model_id = p_model_id AND m.enabled = true;

  RETURN result;
```

```
END;
$$ LANGUAGE plpgsql STABLE;

-- Function: Get workflow config
CREATE OR REPLACE FUNCTION get_workflow_config(p_workflow_id VARCHAR)
RETURNS JSONB AS $$
DECLARE
  result JSONB;
BEGIN
  SELECT jsonb_build_object(
    'id', w.workflow_id,
    'name', w.name,
    'description', w.description,
    'category', w.category,
    'version', w.version,
    'dag', w.dag_definition,
    'inputSchema', w.input_schema,
    'defaultParameters', w.default_parameters,
    'timeout', w.timeout_seconds,
    'minTier', w.min_tier,
    'tasks', (
      SELECT COALESCE(jsonb_agg(jsonb_build_object(
        'taskId', t.task_id, 'name', t.name, 'type', t.task_type,
        'modelId', (SELECT model_id FROM ai_models WHERE id = t.model_id),
        'config', t.config, 'dependsOn', t.depends_on, 'timeout', t.timeout_seconds
      ) ORDER BY t.sequence_order), '[]'::jsonb)
      FROM workflow_tasks t WHERE t.workflow_id = w.id
    ),
    'parameters', (
      SELECT COALESCE(jsonb_agg(jsonb_build_object(
        'name', p.parameter_name, 'displayName', p.display_name,
        'type', p.data_type, 'default', p.default_value,
        'validation', p.validation_rules, 'uiComponent', p.ui_component
      ) ORDER BY p.sequence_order), '[]'::jsonb)
      FROM workflow_parameters p WHERE p.workflow_id = w.id
    )
  ) INTO result
  FROM workflow_definitions w
  WHERE w.workflow_id = p_workflow_id AND w.enabled = true;

  RETURN result;
END;
$$ LANGUAGE plpgsql STABLE;

-- Audit Trigger
CREATE OR REPLACE FUNCTION log_model_changes()
RETURNS TRIGGER AS $$
BEGIN
```

```
  IF TG_OP = 'INSERT' THEN
    INSERT INTO orchestration_audit_log (entity_type, entity_id, action, new_state, change_summ
    VALUES ('model', NEW.id, 'created', to_jsonb(NEW), 'Model ' || NEW.display_name || ' create
  ELSIF TG_OP = 'UPDATE' THEN
    INSERT INTO orchestration_audit_log (entity_type, entity_id, action, previous_state, new_st
    VALUES ('model', NEW.id, 'updated', to_jsonb(OLD), to_jsonb(NEW), 'Model ' || NEW.display_n
  ELSIF TG_OP = 'DELETE' THEN
    INSERT INTO orchestration_audit_log (entity_type, entity_id, action, previous_state, change
    VALUES ('model', OLD.id, 'deleted', to_jsonb(OLD), 'Model ' || OLD.display_name || ' delete
  END IF;
  RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS ai_models_audit_trigger ON ai_models;
CREATE TRIGGER ai_models_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON ai_models
FOR EACH ROW EXECUTE FUNCTION log_model_changes();

-- Row Level Security
ALTER TABLE ai_models ENABLE ROW LEVEL SECURITY;
ALTER TABLE workflow_executions ENABLE ROW LEVEL SECURITY;

CREATE POLICY ai_models_read ON ai_models FOR SELECT TO authenticated
  USING (enabled = true AND status IN ('active', 'beta'));

CREATE POLICY workflow_executions_owner ON workflow_executions FOR ALL TO authenticated
  USING (user_id = auth.uid() OR EXISTS (SELECT 1 FROM administrators WHERE user_id = auth.uid
```

34.2 ALPHAFOLD 2 SEED DATA

packages/database/migrations/034a_seed_alphafold2.sql

```
-- ============================================================================
-- ALPHAFOLD 2 - Nobel Prize-Winning Protein Folding Model
-- ============================================================================

INSERT INTO ai_models (
  model_id, name, display_name, description, category, specialty, provider_type,
  deployment_config, parameters, accuracy, benchmark, capabilities,
  input_formats, output_formats, architecture, performance_metrics,
  thermal_config, pricing_config, min_tier, requires_gpu, gpu_memory_gb,
  status, version, repository, release_date
) VALUES (
  'alphafold2',
  'alphafold2',
  'AlphaFold 2',
```

```
  'Nobel Prize-winning protein structure prediction with near-experimental accuracy. Won CASP14
  'scientific_protein',
  'protein_folding',
  'self_hosted',
  '{
    "image": "pytorch-inference:2.1-gpu-py310-cu121-ubuntu22.04",
    "instanceType": "ml.g5.12xlarge",
    "environment": {"MODEL_NAME": "alphafold2_ptm", "JAX_PLATFORM_NAME": "gpu"},
    "modelDataUrl": "s3://radiant-models/alphafold2/params_2022-12-06.tar",
    "resourceRequirements": {"minGpuMemoryGB": 16, "recommendedGpuMemoryGB": 96, "databaseStora
  }'::JSONB,
  93000000,
  'GDT > 90 on CASP14, 92.4 median GDT_TS, 0.96Å median backbone RMSD',
  'CASP14: Won 88/97 targets, far exceeding all other methods.',
  ARRAY['protein_folding', 'structure_prediction', 'confidence_estimation', 'multimer_predictio
  ARRAY['text/fasta', 'text/plain', 'application/json'],
  ARRAY['application/pdb', 'application/mmcif', 'application/json'],
  '{"evoformerBlocks": 48, "structureBlocks": 8, "recyclingIterations": 4}'::JSONB,
  '{"inferenceTime100Residues": 4.9, "inferenceTime500Residues": 29, "maxResiduesG5_12xlarge":
  '{"defaultState": "OFF", "scaleToZeroAfterMinutes": 10, "warmupTimeSeconds": 180, "minInstanc
  '{"hourlyRate": 14.28, "perRequest": 2.00, "perResidueOver500": 0.002, "markup": 0.75}'::JSO
  4, true, 96, 'active', '2.3.2', 'https://github.com/google-deepmind/alphafold', '2023-04-05'
) ON CONFLICT (model_id) DO UPDATE SET updated_at = NOW();

-- Insert licenses
DO $$
DECLARE alphafold2_id UUID;
BEGIN
  SELECT id INTO alphafold2_id FROM ai_models WHERE model_id = 'alphafold2';

  INSERT INTO model_licenses (model_id, license_type, license_spdx, license_url, commercial_use
  VALUES
    (alphafold2_id, 'source_code', 'Apache-2.0', 'https://github.com/google-deepmind/alphafold/
    (alphafold2_id, 'model_weights', 'CC-BY-4.0', 'https://creativecommons.org/licenses/by/4.0/
    (alphafold2_id, 'database', 'CC-BY-SA-4.0', 'https://creativecommons.org/licenses/by-sa/4.0
  ON CONFLICT DO NOTHING;

  INSERT INTO model_dependencies (model_id, dependency_type, dependency_name, dependency_size_g
  VALUES
    (alphafold2_id, 'database', 'BFD', 1800, 'CC-BY-SA-4.0', true),
    (alphafold2_id, 'database', 'UniRef90', 103, 'CC-BY-4.0', true),
    (alphafold2_id, 'database', 'MGnify', 120, 'CC0-1.0', true),
    (alphafold2_id, 'database', 'PDB70', 56, 'CC0-1.0', true),
    (alphafold2_id, 'database', 'PDB mmCIF', 238, 'CC0-1.0', true)
  ON CONFLICT DO NOTHING;
END $$;

-- Insert protein folding workflow
```

```sql
INSERT INTO workflow_definitions (
  workflow_id, name, description, category, version, dag_definition, input_schema, output_schem
  default_parameters, timeout_seconds, max_retries, min_tier, requires_audit_trail
) VALUES (
  'protein_folding_alphafold2',
  'AlphaFold 2 Protein Folding Pipeline',
  'Complete protein structure prediction using AlphaFold 2 with MSA generation, template search
  'scientific',
  '1.0.0',
  '{"nodes":[{"id":"validate_input","type":"validation"},{"id":"msa_generation","type":"task",
  '{"type":"object","required":["sequence"],"properties":{"sequence":{"type":"string","minLengt
  '{"type":"object","properties":{"structures":{"type":"array"},"confidence":{"type":"object"}}
  '{"numModels":5,"recyclingIterations":4,"relaxStructure":true,"useTemplates":true}'::JSONB,
  7200, 2, 4, true
) ON CONFLICT (workflow_id) DO UPDATE SET updated_at = NOW();
```

34.3 ORCHESTRATION ENGINE SERVICE

packages/services/orchestration/OrchestrationEngine.ts

```typescript
/**
 * RADIANT v4.1.0 - Database-Driven Orchestration Engine
 *
 * KEY PRINCIPLE: ZERO HARDCODING
 * All model configs stored in ai_models table - retrieved via get_model_config()
 */

import { Pool } from 'pg';
import { EventEmitter } from 'events';

export interface ModelConfig {
  id: string;
  name: string;
  displayName: string;
  description?: string;
  category: string;
  providerType: 'self_hosted' | 'external';
  deployment: Record<string, any>;
  thermal: { defaultState: string; scaleToZeroAfterMinutes: number; warmupTimeSeconds: number;
  pricing: { hourlyRate: number; perRequest: number; markup: number };
  parameters: number;
  minTier: number;
  requiresGpu: boolean;
  status: string;
  currentThermalState?: string;
  licenses: Array<{ id: string; type: string; spdx: string; commercialUse: boolean; compliance
  dependencies: Array<{ name: string; type: string; sizeGb?: number; required: boolean }>;
```

```
}

export class OrchestrationEngine extends EventEmitter {
  private pool: Pool;
  private modelCache: Map<string, { config: ModelConfig; timestamp: number }> = new Map();
  private cacheExpiry = 5 * 60 * 1000; // 5 minutes

  constructor(pool: Pool) {
    super();
    this.pool = pool;
  }

  /**
   * Get model config from database - REPLACES hardcoded TypeScript configs
   */
  async getModelConfig(modelId: string): Promise<ModelConfig | null> {
    const cached = this.modelCache.get(modelId);
    if (cached && Date.now() - cached.timestamp < this.cacheExpiry) {
      return cached.config;
    }

    const result = await this.pool.query('SELECT get_model_config($1) as config', [modelId]);
    if (result.rows[0]?.config) {
      const config = result.rows[0].config as ModelConfig;
      this.modelCache.set(modelId, { config, timestamp: Date.now() });
      return config;
    }
    return null;
  }

  /**
   * List models with filters - Used by Admin Dashboard
   */
  async listModels(filters?: { category?: string; status?: string; minTier?: number; providerTy
    let query = `SELECT get_model_config(model_id) as config FROM ai_models WHERE enabled = tru
    const params: any[] = [];
    let idx = 1;

    if (filters?.category) { query += ` AND category = $${idx++}`; params.push(filters.category
    if (filters?.status) { query += ` AND status = $${idx++}`; params.push(filters.status); }
    if (filters?.minTier) { query += ` AND min_tier <= $${idx++}`; params.push(filters.minTier)
    if (filters?.providerType) { query += ` AND provider_type = $${idx++}`; params.push(filters

    query += ` ORDER BY category, display_name`;
    const result = await this.pool.query(query, params);
    return result.rows.map(r => r.config).filter(Boolean);
  }
```

```typescript
/**
 * Create model via Admin Dashboard - NO code deployment needed
 */
async createModel(input: any, adminId: string): Promise<string> {
  const client = await this.pool.connect();
  try {
    await client.query('BEGIN');

    const result = await client.query(`
      INSERT INTO ai_models (
        model_id, name, display_name, description, category, specialty,
        provider_type, deployment_config, parameters, accuracy, benchmark,
        capabilities, input_formats, output_formats, thermal_config, pricing_config,
        min_tier, requires_gpu, gpu_memory_gb, status, version, repository, created_by
      ) VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,$15,$16,$17,$18,$19,$20,$21,$
      RETURNING id
    `, [
      input.modelId, input.name, input.displayName, input.description,
      input.category, input.specialty, input.providerType || 'self_hosted',
      JSON.stringify(input.deployment || {}), input.parameters || 0,
      input.accuracy, input.benchmark, input.capabilities || [],
      input.inputFormats || [], input.outputFormats || [],
      JSON.stringify(input.thermal || {}), JSON.stringify(input.pricing || {}),
      input.minTier || 1, input.requiresGpu || false, input.gpuMemoryGb || 0,
      input.status || 'active', input.version, input.repository, adminId
    ]);

    const modelUuid = result.rows[0].id;

    // Insert licenses
    for (const license of (input.licenses || [])) {
      await client.query(
        `INSERT INTO model_licenses (model_id, license_type, license_spdx, license_url, comme
        [modelUuid, license.type, license.spdx, license.url, license.commercialUse ?? true,
      );
    }

    // Insert dependencies
    for (const dep of (input.dependencies || [])) {
      await client.query(
        `INSERT INTO model_dependencies (model_id, dependency_type, dependency_name, depende
        [modelUuid, dep.type, dep.name, dep.sizeGb, dep.license, dep.required ?? true]
      );
    }

    await client.query('COMMIT');
    this.modelCache.delete(input.modelId);
    this.emit('model:created', { modelId: input.modelId, adminId });
```

```
    return modelUuid;
  } catch (error) {
    await client.query('ROLLBACK');
    throw error;
  } finally {
    client.release();
  }
}

/**
 * Update model via Admin Dashboard
 */
async updateModel(modelId: string, updates: any, adminId: string): Promise<void> {
  const sets: string[] = [];
  const params: any[] = [];
  let idx = 1;

  const fields: Record<string, string> = {
    displayName: 'display_name', description: 'description', category: 'category',
    status: 'status', minTier: 'min_tier', version: 'version'
  };

  for (const [key, col] of Object.entries(fields)) {
    if (updates[key] !== undefined) { sets.push(`${col} = $${idx++}`); params.push(updates[ke
  }

  if (updates.deployment) { sets.push(`deployment_config = $${idx++}`); params.push(JSON.stri
  if (updates.thermal) { sets.push(`thermal_config = $${idx++}`); params.push(JSON.stringify
  if (updates.pricing) { sets.push(`pricing_config = $${idx++}`); params.push(JSON.stringify

  if (sets.length === 0) return;

  sets.push(`updated_at = NOW()`, `updated_by = $${idx++}`);
  params.push(adminId, modelId);

  await this.pool.query(`UPDATE ai_models SET ${sets.join(', ')} WHERE model_id = $${idx}`, p
  this.modelCache.delete(modelId);
  this.emit('model:updated', { modelId, adminId });
}

/**
 * Delete model (soft delete)
 */
async deleteModel(modelId: string, adminId: string): Promise<void> {
  await this.pool.query(
    `UPDATE ai_models SET enabled = false, status = 'disabled', updated_at = NOW(), updated_b
    [modelId, adminId]
  );
```

```
  this.modelCache.delete(modelId);
  this.emit('model:deleted', { modelId, adminId });
}

/**
 * Get workflow config from database
 */
async getWorkflowConfig(workflowId: string): Promise<any> {
  const result = await this.pool.query('SELECT get_workflow_config($1) as config', [workflow
  return result.rows[0]?.config || null;
}

/**
 * Execute workflow with parameters
 */
async executeWorkflow(workflowId: string, tenantId: string, userId: string, parameters: Reco
  const workflow = await this.getWorkflowConfig(workflowId);
  if (!workflow) throw new Error(`Workflow not found: ${workflowId}`);

  const resolvedParams = { ...workflow.defaultParameters, ...parameters };

  const result = await this.pool.query(`
    INSERT INTO workflow_executions (workflow_id, tenant_id, user_id, status, input_paramete
    VALUES ((SELECT id FROM workflow_definitions WHERE workflow_id = $1), $2, $3, 'running',
    RETURNING id
  `, [workflowId, tenantId, userId, JSON.stringify(parameters), JSON.stringify(resolvedParams

  const executionId = result.rows[0].id;
  this.emit('workflow:started', { executionId, workflowId, tenantId, userId });
  return executionId;
}

/**
 * Get license summary for compliance dashboard
 */
async getLicenseSummary(): Promise<{ totalModels: number; commercialOk: number; reviewNeeded
  const result = await this.pool.query(`
    SELECT
      COUNT(DISTINCT m.id) as total_models,
      COUNT(DISTINCT CASE WHEN NOT EXISTS (SELECT 1 FROM model_licenses l2 WHERE l2.model_id
      COUNT(DISTINCT CASE WHEN l.compliance_status = 'review_needed' THEN m.id END) as revie
      COUNT(DISTINCT CASE WHEN l.compliance_status = 'non_compliant' THEN m.id END) as non_c
      COUNT(DISTINCT CASE WHEN l.expires_at IS NOT NULL AND l.expires_at < NOW() + INTERVAL
    FROM ai_models m LEFT JOIN model_licenses l ON m.id = l.model_id WHERE m.enabled = true
  `);
  return {
    totalModels: parseInt(result.rows[0].total_models),
    commercialOk: parseInt(result.rows[0].commercial_ok),
```

```
      reviewNeeded: parseInt(result.rows[0].review_needed),
      nonCompliant: parseInt(result.rows[0].non_compliant),
      expiringIn30Days: parseInt(result.rows[0].expiring_soon),
    };
  }

  refreshCache(): void {
    this.modelCache.clear();
  }
}

let instance: OrchestrationEngine | null = null;
export function getOrchestrationEngine(pool: Pool): OrchestrationEngine {
  if (!instance) instance = new OrchestrationEngine(pool);
  return instance;
}
```
