Contents

SECTION 13: USER NEURAL ENGINE (v3.0.0)

13.1 Neural Engine Overview

The User Neural Engine provides personalized AI experiences through learned preferences, conversation memory with embeddings, and adaptive behavior patterns.

13.2 Neural Engine Database Schema

```sql
-- migrations/023_user_neural_engine.sql

-- Enable pgvector for embeddings
CREATE EXTENSION IF NOT EXISTS vector;

CREATE TABLE user_preferences (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    preference_key VARCHAR(100) NOT NULL,
    preference_value JSONB NOT NULL,
    confidence DECIMAL(3, 2) DEFAULT 0.5,
    learned_from TEXT[],
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(tenant_id, user_id, preference_key)
);

CREATE TABLE user_memory (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    memory_type VARCHAR(50) NOT NULL,
    content TEXT NOT NULL,
    embedding vector(1536),
    importance DECIMAL(3, 2) DEFAULT 0.5,
    access_count INTEGER DEFAULT 0,
    last_accessed TIMESTAMPTZ,
```

```sql
    expires_at TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE user_behavior_patterns (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    pattern_type VARCHAR(50) NOT NULL,
    pattern_data JSONB NOT NULL,
    occurrence_count INTEGER DEFAULT 1,
    last_occurred TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_user_memory_embedding ON user_memory USING ivfflat (embedding vector_cosine_op
CREATE INDEX idx_user_preferences_user ON user_preferences(tenant_id, user_id);
CREATE INDEX idx_user_patterns_user ON user_behavior_patterns(tenant_id, user_id);

ALTER TABLE user_preferences ENABLE ROW LEVEL SECURITY;
ALTER TABLE user_memory ENABLE ROW LEVEL SECURITY;
ALTER TABLE user_behavior_patterns ENABLE ROW LEVEL SECURITY;

CREATE POLICY user_preferences_isolation ON user_preferences USING (tenant_id = current_setting
CREATE POLICY user_memory_isolation ON user_memory USING (tenant_id = current_setting('app.cur
CREATE POLICY user_patterns_isolation ON user_behavior_patterns USING (tenant_id = current_set
```

13.3 Neural Engine Service

```typescript
// packages/core/src/services/neural-engine.ts

import { Pool } from 'pg';
import { BedrockRuntimeClient, InvokeModelCommand } from '@aws-sdk/client-bedrock-runtime';

export class NeuralEngine {
    private pool: Pool;
    private bedrock: BedrockRuntimeClient;

    constructor(pool: Pool) {
        this.pool = pool;
        this.bedrock = new BedrockRuntimeClient({});
    }

    async learnFromConversation(
        tenantId: string,
        userId: string,
        messages: { role: string; content: string }[]
    ): Promise<void> {
```

```
        // Extract preferences
        const preferences = await this.extractPreferences(messages);
        for (const pref of preferences) {
            await this.updatePreference(tenantId, userId, pref.key, pref.value, pref.confidenc
        }

        // Store important memories
        const memories = await this.extractMemories(messages);
        for (const memory of memories) {
            await this.storeMemory(tenantId, userId, memory);
        }

        // Update behavior patterns
        await this.updateBehaviorPatterns(tenantId, userId, messages);
    }

    async getRelevantMemories(
        tenantId: string,
        userId: string,
        query: string,
        limit: number = 5
    ): Promise<any[]> {
        const embedding = await this.generateEmbedding(query);

        const result = await this.pool.query(`
            SELECT content, importance, memory_type,
                   1 - (embedding <=> $4::vector) as similarity
            FROM user_memory
            WHERE tenant_id = $1 AND user_id = $2
            AND (expires_at IS NULL OR expires_at > NOW())
            ORDER BY embedding <=> $4::vector
            LIMIT $3
        `, [tenantId, userId, limit, `[${embedding.join(',')}]`]);

        // Update access counts
        for (const row of result.rows) {
            await this.pool.query(`
                UPDATE user_memory SET access_count = access_count + 1, last_accessed = NOW()
                WHERE id = $1
            `, [row.id]);
        }

        return result.rows;
    }

    async getPreferences(tenantId: string, userId: string): Promise<Record<string, any>> {
        const result = await this.pool.query(`
            SELECT preference_key, preference_value, confidence
```

```
            FROM user_preferences
            WHERE tenant_id = $1 AND user_id = $2
            ORDER BY confidence DESC
        `, [tenantId, userId]);

        const prefs: Record<string, any> = {};
        for (const row of result.rows) {
            prefs[row.preference_key] = {
                value: row.preference_value,
                confidence: row.confidence
            };
        }
        return prefs;
    }

    private async generateEmbedding(text: string): Promise<number[]> {
        const response = await this.bedrock.send(new InvokeModelCommand({
            modelId: 'amazon.titan-embed-text-v1',
            body: JSON.stringify({ inputText: text }),
            contentType: 'application/json'
        }));

        const result = JSON.parse(new TextDecoder().decode(response.body));
        return result.embedding;
    }

    private async extractPreferences(messages: any[]): Promise<any[]> {
        const conversationText = messages.map(m => `${m.role}: ${m.content}`).join('\n');

        const response = await this.bedrock.send(new InvokeModelCommand({
            modelId: 'anthropic.claude-3-haiku-20240307-v1:0',
            body: JSON.stringify({
                anthropic_version: 'bedrock-2023-05-31',
                max_tokens: 1024,
                messages: [{
                    role: 'user',
                    content: `Extract user preferences from this conversation. Return JSON arr

${conversationText}

Return only valid JSON array.`
                }]
            }),
            contentType: 'application/json'
        }));

        const result = JSON.parse(new TextDecoder().decode(response.body));
        try {
```

```
            return JSON.parse(result.content[0].text);
        } catch {
            return [];
        }
    }

    private async extractMemories(messages: any[]): Promise<any[]> {
        // Similar extraction for important facts/memories
        return [];
    }

    private async updatePreference(
        tenantId: string,
        userId: string,
        key: string,
        value: any,
        confidence: number
    ): Promise<void> {
        await this.pool.query(`
            INSERT INTO user_preferences (tenant_id, user_id, preference_key, preference_value
            VALUES ($1, $2, $3, $4, $5)
            ON CONFLICT (tenant_id, user_id, preference_key)
            DO UPDATE SET
                preference_value = EXCLUDED.preference_value,
                confidence = GREATEST(user_preferences.confidence, EXCLUDED.confidence),
                updated_at = NOW()
        `, [tenantId, userId, key, JSON.stringify(value), confidence]);
    }

    private async storeMemory(tenantId: string, userId: string, memory: any): Promise<void> {
        const embedding = await this.generateEmbedding(memory.content);

        await this.pool.query(`
            INSERT INTO user_memory (tenant_id, user_id, memory_type, content, embedding, impo
            VALUES ($1, $2, $3, $4, $5::vector, $6)
        `, [tenantId, userId, memory.type, memory.content, `[${embedding.join(',')}]`, memory.
    }

    private async updateBehaviorPatterns(
        tenantId: string,
        userId: string,
        messages: any[]
    ): Promise<void> {
        // Pattern detection logic
    }
}
```