Claude Code Neurales Integrationsframework mit RAG und Embeddings

<pattern_recognition>
Dieses Framework transformiert Claude Code von einem reinen Entwicklungstool zu einem Neural Agent Framework mit erweiterten Kontext-Fähigkeiten durch RAG-Integration (Retrieval-Augmented Generation) und Embedding-Technologie. Es verändert die Art, wie künstliche Intelligenz mit Codebases und Wissensbasen interagiert, indem es die Grenzen zwischen menschlicher Expertise und KI-Unterstützung verwischt.
</pattern_recognition>

Übersicht

Das erweiterte Framework baut auf der vorhandenen Claude Code-Struktur auf und integriert:

1. **Leichtgewichtiges RAG-System**: Semantische Suche in der eigenen Codebasis und Dokumentation
2. **Embedding-Integration**: Vector Embedding für Codeteile, Dokumentation und Kontext
3. **VibeCodingFramework-Anbindung**: Direkte Integration mit Next.js 15, Tailwind CSS 4, etc.
4. **Neural Agent Capabilities**: Der Benutzer wird als "Agent" in das System eingebunden

Dieses Dokument beschreibt die Implementation, Struktur und Nutzung des erweiterten Frameworks.

Erweiterte Ordnerstruktur

Aufbauend auf der vorhandenen Struktur im `/home/jan/claude-code`-Verzeichnis:

```
/home/jan/claude-code/
├── .claude/                    # Prozedurales Gedächtnis (bereits vorhanden)
│   ├── CLAUDE.md               # Meta-Framework (bereits vorhanden)
│   ├── commands/               # Befehle (erweitert)
│   │   ├── analyze-complexity.md     # (bereits vorhanden)
│   │   ├── embed-document.md         # (neu: Embedding-Befehl)
│   │   ├── rag-query.md              # (neu: RAG-Abfrage-Befehl)
│   │   └── train-embeddings.md       # (neu: Embedding-Training-Befehl)
│   ├── scripts/                # (bereits vorhanden)
│   ├── config/                 # (erweitert)
│   │   ├── rag.json            # (neu: RAG-Konfiguration)
│   │   └── embeddings.json     # (neu: Embedding-Konfiguration)
│   └── tools/                  # (neu: Tool-Integration)
│       ├── rag/                # RAG-Tools
│       └── embeddings/         # Embedding-Tools
├── .clauderules                # (bereits vorhanden, erweitert)
├── .mcp.json                   # (bereits vorhanden, erweitert)
├── ai_docs/                    # Episodisches Gedächtnis (bereits vorhanden)
│   ├── examples/               # (bereits vorhanden, erweitert)
│   │   ├── rag-query.md        # (neu: RAG-Abfrage-Beispiel)
│   │   └── embedding-example.md # (neu: Embedding-Beispiel)
│   ├── prompts/                # (bereits vorhanden, erweitert)
│   │   ├── rag-prompts/        # (neu: RAG-spezifische Prompts)
│   │   └── embedding-prompts/ # (neu: Embedding-spezifische Prompts)
│   ├── templates/              # (bereits vorhanden, erweitert)
│   ├── embedding/              # (neu: Embedding-Dokumentation)
│   │   ├── README.md           # Einführung in Embeddings
│   │   ├── models/             # Dokumentation zu Embedding-Modellen
│   │   └── integration/        # Integrationsleitfäden
│   └── rag/                    # (neu: RAG-Dokumentation)
│       ├── README.md           # Einführung in RAG
│       ├── vector-stores/      # Dokumentation zu Vektorenspeichern
│       └── strategies/         # RAG-Strategien und Best Practices
├── specs/                      # Semantisches Gedächtnis (bereits vorhanden)
│   ├── schemas/                # (bereits vorhanden, erweitert)
│   │   ├── api-schema.json     # (bereits vorhanden)
│   │   ├── rag-schema.json     # (neu: RAG-Konfigurationsschema)
│   │   └── embedding-schema.json # (neu: Embedding-Konfigurationsschema)
│   ├── openapi/                # (bereits vorhanden)
│   ├── migrations/             # (bereits vorhanden)
│   ├── embeddings/             # (neu: Embedding-Spezifikationen)
│   │   ├── voyage.md           # Voyage AI Embedding-Spezifikation
│   │   └── huggingface.md      # Hugging Face Embedding-Spezifikation
│   └── integrations/           # (neu: Integrationsspezifikationen)
│       └── vibecodingframework.md # Spezifikation für VibeCodingFramework
└── integration/                # (neu: Integrationskomponenten)
```

```
├── vibecodingframework/  # VibeCodingFramework-Integration
│   ├── README.md         # Dokumentation
│   ├── api/              # API-Routes für Next.js
│   ├── components/       # React-Komponenten
│   └── hooks/            # React-Hooks
├── nextjs/               # Next.js-spezifische Integrationen
└── database/             # Datenbankadapter
    ├── supabase.js       # Supabase-Adapter
    └── sqlite.js         # SQLite-Adapter
```

RAG-System Implementierung

1. Konfiguration (/.claude/config/rag.json)

json

```json
{
  "database": {
    "type": "lancedb",
    "connection": {
      "path": "data/lancedb"
    }
  },
  "embedding": {
    "provider": "voyage",
    "model": "voyage-2",
    "dimensions": 1024,
    "api_key_env": "VOYAGE_API_KEY"
  },
  "retrieval": {
    "top_k": 5,
    "similarity_threshold": 0.7,
    "reranking": false
  },
  "cache": {
    "enabled": true,
    "ttl": 3600,
    "strategy": "lru"
  }
}
```

2. Schema Definition (specs/schemas/rag-schema.json)

json

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "RAG System Configuration",
  "type": "object",
  "properties": {
    "database": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string",
          "enum": ["lancedb", "chromadb", "postgres", "sqlite"]
        },
        "connection": {
          "type": "object",
          "properties": {
            "path": { "type": "string" },
            "url": { "type": "string" },
            "credentials": { "type": "object" }
          }
        }
      },
      "required": ["type"]
    },
    "embedding": {
      "type": "object",
      "properties": {
        "provider": {
          "type": "string",
          "enum": ["voyage", "openai", "huggingface", "cohere"]
        },
        "model": { "type": "string" },
        "dimensions": { "type": "integer" },
        "api_key_env": { "type": "string" }
      },
      "required": ["provider", "model"]
    },
    "retrieval": {
      "type": "object",
      "properties": {
        "top_k": { "type": "integer", "default": 5 },
        "similarity_threshold": { "type": "number", "default": 0.7 },
        "reranking": { "type": "boolean", "default": false }
      }
    },
    "cache": {
      "type": "object",
      "properties": {
        "enabled": { "type": "boolean", "default": true },
        "ttl": { "type": "integer", "default": 3600 },
        "strategy": {
          "type": "string",
          "enum": ["lru", "fifo", "lfu"],
          "default": "lru"
        }
      }
    }
  },
  "required": ["database", "embedding", "retrieval"]
}
```

3. Embedding-Befehl (.claude/commands/embed-document.md)

```markdown
# Embed Document

Analyze and embed a document into the vector database for future RAG usage.

## Usage
/embed-document $ARGUMENTS

## Parameters
- path: Path to the file or directory to embed
- namespace: (Optional) Namespace to store the embeddings
- chunk_size: (Optional) Size of text chunks for embedding (default: 1000)
- overlap: (Optional) Overlap between chunks (default: 200)

## Example
/embed-document path=specs/schemas/rag-schema.json namespace=schemas

The command will:
1. Read and preprocess the document
2. Split into optimal chunks based on content
3. Generate embeddings using the configured provider
4. Store in the vector database
5. Create metadata for retrieval

Results include document ID and verification of successful embedding.
```

4. RAG-Abfrage-Beispiel (ai_docs/examples/rag-query.md)

markdown

~~~markdown
# RAG Query Example

This example demonstrates how to query the RAG system using Claude integration.

```python
import os
from claude_code_rag import RagClient, ClaudeIntegration

# Initialize RAG client with configuration
rag_client = RagClient(config_path=".claude/config/rag.json")

# Initialize Claude integration
claude = ClaudeIntegration(api_key=os.environ["CLAUDE_API_KEY"])

# Define a query
query = "How does the vector database integration work with Claude?"

# Retrieve relevant context
contexts = rag_client.query(query, top_k=3)

# Format context for Claude
context_text = "\n\n".join([ctx.text for ctx in contexts])

# Create a prompt with retrieved context
prompt = f"""
You are an assistant that answers questions based on the provided context.

Context:
{context_text}

Question: {query}

Please answer the question based only on the provided context. If the context doesn'
"""

# Get response from Claude
response = claude.complete(prompt)

print(response)
~~~

This example uses the RAG client to retrieve relevant documents based on the query, then sends those documents as context to Claude for generating a response.

~~~
## Embedding-Integration

### 1. Konfiguration (/.claude/config/embeddings.json)

```json
{
  "providers": {
    "voyage": {
      "api_key_env": "VOYAGE_API_KEY",
      "default_model": "voyage-2",
      "dimensions": 1024,
      "batch_size": 32
    },
    "huggingface": {
      "model": "sentence-transformers/all-mpnet-base-v2",
      "device": "cpu",
      "dimensions": 768,
      "batch_size": 16
    }
  },
  "default_provider": "voyage",
  "cache": {
    "enabled": true,
    "storage": "file",
    "path": ".cache/embeddings",
    "ttl": 86400
  },
  "chunking": {
    "strategy": "semantic",
    "size": 1000,
    "overlap": 200
  }
}
~~~

2. Schema Definition (specs/schemas/embedding-schema.json)

json

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Embedding Configuration",
  "type": "object",
  "properties": {
    "providers": {
      "type": "object",
      "additionalProperties": {
        "type": "object",
        "properties": {
          "api_key_env": { "type": "string" },
          "default_model": { "type": "string" },
          "dimensions": { "type": "integer" },
          "batch_size": { "type": "integer" }
        }
      }
    },
    "default_provider": { "type": "string" },
    "cache": {
      "type": "object",
      "properties": {
        "enabled": { "type": "boolean" },
        "storage": { "type": "string", "enum": ["file", "memory", "redis"] },
        "path": { "type": "string" },
        "ttl": { "type": "integer" }
      }
    },
    "chunking": {
      "type": "object",
      "properties": {
        "strategy": { "type": "string", "enum": ["semantic", "fixed", "sentence", "pa
        "size": { "type": "integer" },
        "overlap": { "type": "integer" }
      }
    }
  },
  "required": ["providers", "default_provider"]
}
```

VibeCodingFramework-Integration

1. Integration-Readme (integration/vibecodingframework/README.md)

markdown

```
# Claude Code Integration with VibeCodingFramework

This directory contains the necessary integration components to connect Claude Code w

## Components

### API Integration

The `/api/claude` directory contains API routes that can be added to your Next.js ap

- `/api/claude/embed` - For generating and storing embeddings
- `/api/claude/query` - For querying the RAG system
- `/api/claude/chat` - For interacting with Claude with RAG-enhanced context

### React Components

- `ClaudeProvider` - Context provider for Claude integration
- `RagSearch` - Search component for querying the RAG system
- `EmbeddingManager` - Component for managing embeddings
- `AgentContext` - Component for displaying agent context from `.about` profiles

### Database Adapters

- `SupabaseAdapter` - For using Supabase as vector database
- `SQLiteAdapter` - For using SQLite as vector database (with sqlite-vss extension)

## Installation

1. Copy the integration directory to your VibeCodingFramework project
2. Install dependencies: `npm install @anthropic/sdk lancedb chromadb`
3. Configure `.env`:
```

CLAUDE_API_KEY=your_api_key
VOYAGE_API_KEY=your_voyage_api_key (if using Voyage embeddings)
DB_TYPE=supabase|sqlite

```
4. Import components as needed in your application
```

2. Supabase-Adapter (integration/database/supabase.js)

```
javascript
```

```
/**
 * Supabase adapter for the Claude Code RAG system
 * Use with Supabase's pgvector extension
 */

const { createClient } = require('@supabase/supabase-js');

class SupabaseAdapter {
  constructor(config) {
    const supabaseUrl = process.env.SUPABASE_URL;
    const supabaseKey = process.env.SUPABASE_KEY;

    if (!supabaseUrl || !supabaseKey) {
      throw new Error('SUPABASE_URL and SUPABASE_KEY environment variables are requi
    }

    this.client = createClient(supabaseUrl, supabaseKey);
    this.tableName = config.tableName || 'embeddings';
    this.dimensions = config.dimensions || 1024;

    this.initialized = false;
  }

  async initialize() {
    // Check if the table exists, if not create it
    const { error } = await this.client.rpc('create_embeddings_table', {
      table_name: this.tableName,
      dimensions: this.dimensions
    });

    if (error && !error.message.includes('already exists')) {
      throw error;
    }

    this.initialized = true;
  }

  async addEmbedding(id, vector, metadata = {}, namespace = 'default') {
    if (!this.initialized) await this.initialize();

    const { error } = await this.client
      .from(this.tableName)
      .insert({
        id,
        embedding: vector,
        metadata: JSON.stringify(metadata),
      namespace,
      created_at: new Date().toISOString()
    });

  if (error) throw error;

  return { id };
}

async search(queryVector, options = {}) {
  if (!this.initialized) await this.initialize();

  const {
    top_k = 5,
    namespace = 'default',
    threshold = 0.7
  } = options;

  const { data, error } = await this.client.rpc('match_embeddings', {
    query_embedding: queryVector,
    match_threshold: threshold,
    match_count: top_k,
    filter_namespace: namespace
  });

  if (error) throw error;

  return data.map(item => ({
    id: item.id,
    score: item.similarity,
    metadata: JSON.parse(item.metadata)
  }));
}

async deleteEmbedding(id) {
  if (!this.initialized) await this.initialize();

  const { error } = await this.client
    .from(this.tableName)
    .delete()
    .eq('id', id);

  if (error) throw error;

  return { id };
}
```

```
    async deleteNamespace(namespace) {
      if (!this.initialized) await this.initialize();

      const { error } = await this.client
        .from(this.tableName)
        .delete()
        .eq('namespace', namespace);

      if (error) throw error;

      return { namespace };
    }
  }

  module.exports = SupabaseAdapter;
```

3. SQLite-Adapter (integration/database/sqlite.js)

```
javascript
```

```
/**
 * SQLite adapter for the Claude Code RAG system
 * Uses sqlite-vss extension for vector search
 */

const sqlite3 = require('sqlite3');
const { open } = require('sqlite');
const path = require('path');
const fs = require('fs');

class SQLiteAdapter {
  constructor(config) {
    this.dbPath = config.path || path.join(process.cwd(), 'data', 'sqlite', 'embeddi
    this.dimensions = config.dimensions || 1024;
    this.initialized = false;
    this.db = null;
  }

  async initialize() {
    // Ensure directory exists
    const dir = path.dirname(this.dbPath);
    if (!fs.existsSync(dir)) {
      fs.mkdirSync(dir, { recursive: true });
    }

    // Open database connection
    this.db = await open({
      filename: this.dbPath,
      driver: sqlite3.Database
    });

    // Load VSS extension
    await this.db.exec(`LOAD EXTENSION 'sqlite_vss'`);

    // Create tables if they don't exist
    await this.db.exec(`
      CREATE TABLE IF NOT EXISTS embeddings (
        id TEXT PRIMARY KEY,
        namespace TEXT NOT NULL,
        metadata TEXT,
        created_at TEXT NOT NULL
      );

      CREATE VIRTUAL TABLE IF NOT EXISTS embedding_vectors USING vss0(
        embedding(${this.dimensions}),
        id,
      distance_function(cosine)
    );
  `);

  // Create indexes
  await this.db.exec(`
    CREATE INDEX IF NOT EXISTS idx_namespace ON embeddings(namespace);
    CREATE INDEX IF NOT EXISTS idx_created_at ON embeddings(created_at);
  `);

  this.initialized = true;
}

async addEmbedding(id, vector, metadata = {}, namespace = 'default') {
  if (!this.initialized) await this.initialize();

  // Begin transaction
  await this.db.exec('BEGIN TRANSACTION');

  try {
    // Add to embeddings table
    await this.db.run(
      `INSERT OR REPLACE INTO embeddings (id, namespace, metadata, created_at)
       VALUES (?, ?, ?, ?)`,
      [id, namespace, JSON.stringify(metadata), new Date().toISOString()]
    );

    // Add to vector table
    await this.db.run(
      `INSERT OR REPLACE INTO embedding_vectors (id, embedding)
       VALUES (?, ?)`,
      [id, JSON.stringify(vector)]
    );

    // Commit transaction
    await this.db.exec('COMMIT');

    return { id };
  } catch (error) {
    // Rollback transaction
    await this.db.exec('ROLLBACK');
    throw error;
  }
}

async search(queryVector, options = {}) {
  if (!this.initialized) await this.initialize();
```

```
    const {
      top_k = 5,
      namespace = 'default',
      threshold = 0.7
    } = options;

    // Convert threshold to distance (cosine similarity to distance)
    const distance = 1 - threshold;

    const results = await this.db.all(`
      SELECT v.id, e.metadata, 1 - distance AS similarity
      FROM embedding_vectors v
      JOIN embeddings e ON v.id = e.id
      WHERE e.namespace = ?
      AND vss_search(
        embedding,
        ?,
        ?
      ) <= ?
      LIMIT ?
    `, [namespace, JSON.stringify(queryVector), 'cosine', distance, top_k]);

    return results.map(row => ({
      id: row.id,
      score: row.similarity,
      metadata: JSON.parse(row.metadata)
    }));
  }

  async deleteEmbedding(id) {
    if (!this.initialized) await this.initialize();

    await this.db.exec('BEGIN TRANSACTION');

    try {
      await this.db.run('DELETE FROM embeddings WHERE id = ?', [id]);
      await this.db.run('DELETE FROM embedding_vectors WHERE id = ?', [id]);

      await this.db.exec('COMMIT');

      return { id };
    } catch (error) {
      await this.db.exec('ROLLBACK');
      throw error;
    }
  }
  async deleteNamespace(namespace) {
    if (!this.initialized) await this.initialize();

    // Get all IDs in namespace
    const rows = await this.db.all(
      'SELECT id FROM embeddings WHERE namespace = ?',
      [namespace]
    );

    // Delete each embedding
    await this.db.exec('BEGIN TRANSACTION');

    try {
      for (const row of rows) {
        await this.db.run('DELETE FROM embedding_vectors WHERE id = ?', [row.id]);
      }

      await this.db.run('DELETE FROM embeddings WHERE namespace = ?', [namespace]);

      await this.db.exec('COMMIT');

      return { namespace, count: rows.length };
    } catch (error) {
      await this.db.exec('ROLLBACK');
      throw error;
    }
  }

  async close() {
    if (this.db) {
      await this.db.close();
      this.initialized = false;
    }
  }
}

module.exports = SQLiteAdapter;
```

Installations-Script (bin/setup-rag.sh)

bash

```bash
#!/bin/bash

# Setup script for Claude Code RAG system
# This script installs the necessary dependencies and configures the RAG system

set -e

NC='\033[0m'
BLUE='\033[0;34m'
GREEN='\033[0;32m'
YELLOW='\033[0;33m'
RED='\033[0;31m'

info() {
  echo -e "${BLUE}[INFO]${NC} $1"
}

success() {
  echo -e "${GREEN}[SUCCESS]${NC} $1"
}

warn() {
  echo -e "${YELLOW}[WARNING]${NC} $1"
}

error() {
  echo -e "${RED}[ERROR]${NC} $1"
  exit 1
}

# Check if Python is installed
info "Checking Python installation..."
if ! command -v python3 &> /dev/null; then
  error "Python 3 is not installed. Please install Python 3.8 or higher."
fi

# Create virtual environment
info "Creating virtual environment..."
python3 -m venv .venv
source .venv/bin/activate

# Install Python dependencies
info "Installing Python dependencies..."
pip install -U pip
pip install langchain lancedb chromadb anthropic sentence-transformers voyage-embedd
```

```
# Setup database
info "Setting up vector database..."
if [ -f ".env" ]; then
  source .env
fi

DB_TYPE=${DB_TYPE:-"lancedb"}
if [ "$DB_TYPE" = "lancedb" ]; then
  info "Setting up LanceDB..."
  mkdir -p data/lancedb
elif [ "$DB_TYPE" = "chromadb" ]; then
  info "Setting up ChromaDB..."
  mkdir -p data/chromadb
elif [ "$DB_TYPE" = "supabase" ]; then
  info "Setting up Supabase vector store..."
  if [ -z "$SUPABASE_URL" ] || [ -z "$SUPABASE_KEY" ]; then
    warn "Supabase URL or key not found in .env. You will need to configure them man
  fi
else
  warn "Unknown database type: $DB_TYPE. Defaulting to LanceDB."
  mkdir -p data/lancedb
  DB_TYPE="lancedb"
fi

# Create config files
info "Creating configuration files..."
mkdir -p .claude/config

cat > .claude/config/rag.json << EOF
{
  "database": {
    "type": "${DB_TYPE}",
    "connection": {
      "path": "data/${DB_TYPE}"
    }
  },
  "embedding": {
    "provider": "voyage",
    "model": "voyage-2",
    "dimensions": 1024,
    "api_key_env": "VOYAGE_API_KEY"
  },
  "retrieval": {
    "top_k": 5,
    "similarity_threshold": 0.7,
    "reranking": false
  },
  "cache": {
    "enabled": true,
    "ttl": 3600,
    "strategy": "lru"
  }
}
EOF

success "RAG system setup complete!"
success "You can nun Claude Code mit RAG-Unterstützung verwenden."
info "Um die Umgebung zu aktivieren: source .venv/bin/activate"
info "Um Dokumente zu embedden: claude-code /embed-document path=your/file.md"
info "Um das RAG-System abzufragen: claude-code \"Query with RAG context\""
```

Integration mit dem User-Agent-System (vibecodingframework)

Das Framework unterstützt die Transformation von Benutzern zu "Agenten" im System, indem es .about-Profile für jeden Benutzer erstellt und verwaltet. Diese Profile werden für die Personalisierung von RAG-Ergebnissen und Claude-Interaktionen verwendet.

Agent-Profil-Schema (specs/schemas/agent-profile-schema.json)

json

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Agent Profile Configuration",
  "type": "object",
  "properties": {
    "user_id": { "type": "string" },
    "agent_state": {
      "type": "string",
      "enum": ["active", "inactive", "learning"]
    },
    "name": { "type": "string" },
    "goals": {
      "type": "array",
      "items": { "type": "string" }
    },
    "companies": {
      "type": "array",
      "items": { "type": "string" }
    },
    "preferences": {
      "type": "object",
      "properties": {
        "theme": { "type": "string" },
        "language": { "type": "string" }
      }
    },
    "is_agent": { "type": "boolean" },
    "created_at": { "type": "string", "format": "date-time" },
    "updated_at": { "type": "string", "format": "date-time" }
  },
  "required": ["user_id", "agent_state", "is_agent"]
}
```

Agent-Profil-Erstellung (integration/vibecodingframework/components/AgentProfileForm.jsx)

```
jsx
```

```
import { useState } from 'react';
import { Button, Input, Textarea, Switch, Card, CardHeader, CardContent, CardFooter

export default function AgentProfileForm({ onSubmit, initialData = {} }) {
  const [form, setForm] = useState({
    name: initialData.name || '',
    goals: initialData.goals?.join('\n') || '',
    companies: initialData.companies?.join('\n') || '',
    preferences: {
      theme: initialData.preferences?.theme || 'system',
      language: initialData.preferences?.language || 'en'
    },
    is_agent: initialData.is_agent || false,
    ...initialData
  });

  const handleChange = (field, value) => {
    setForm(prev => ({
      ...prev,
      [field]: value
    }));
  };

  const handlePreferenceChange = (field, value) => {
    setForm(prev => ({
      ...prev,
      preferences: {
        ...prev.preferences,
        [field]: value
      }
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    // Format data for submission
    const formattedData = {
      ...form,
      goals: form.goals.split('\n').filter(Boolean),
      companies: form.companies.split('\n').filter(Boolean),
      agent_state: form.is_agent ? 'active' : 'inactive',
      updated_at: new Date().toISOString()
    };

    if (!formattedData.created_at) {
        formattedData.created_at = new Date().toISOString();
      }

      onSubmit(formattedData);
    };

    return (
      <Card className="w-full max-w-2xl">
        <CardHeader>
          <h2 className="text-2xl font-bold">Agent Profile</h2>
          <p className="text-gray-500">Create or update your agent profile</p>
        </CardHeader>

        <CardContent>
          <form onSubmit={handleSubmit} className="space-y-4">
            <div>
              <label htmlFor="name" className="block text-sm font-medium">Name</label>
              <Input
                id="name"
                value={form.name}
                onChange={(e) => handleChange('name', e.target.value)}
                placeholder="Your name"
              />
            </div>

            <div>
              <label htmlFor="goals" className="block text-sm font-medium">Goals (one
              <Textarea
                id="goals"
                value={form.goals}
                onChange={(e) => handleChange('goals', e.target.value)}
                placeholder="Your goals, one per line"
                rows={4}
              />
            </div>

            <div>
              <label htmlFor="companies" className="block text-sm font-medium">Compani
              <Textarea
                id="companies"
                value={form.companies}
                onChange={(e) => handleChange('companies', e.target.value)}
                placeholder="Your companies, one per line"
                rows={2}
              />
            </div>
          <div className="grid grid-cols-2 gap-4">
            <div>
              <label htmlFor="theme" className="block text-sm font-medium">Theme</la
              <select
                id="theme"
                value={form.preferences.theme}
                onChange={(e) => handlePreferenceChange('theme', e.target.value)}
                className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
              >
                <option value="system">System</option>
                <option value="light">Light</option>
                <option value="dark">Dark</option>
              </select>
            </div>

            <div>
              <label htmlFor="language" className="block text-sm font-medium">Langua
              <select
                id="language"
                value={form.preferences.language}
                onChange={(e) => handlePreferenceChange('language', e.target.value)}
                className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
              >
                <option value="en">English</option>
                <option value="de">Deutsch</option>
                <option value="fr">Français</option>
                <option value="es">Español</option>
              </select>
            </div>
          </div>

          <div className="flex items-center justify-between">
            <span className="text-sm font-medium">Become an AI-connected agent</span
            <Switch
              checked={form.is_agent}
              onCheckedChange={(checked) => handleChange('is_agent', checked)}
            />
          </div>
        </form>
      </CardContent>

      <CardFooter>
        <Button type="submit" onClick={handleSubmit}>
          {initialData.user_id ? 'Update Profile' : 'Create Profile'}
        </Button>
      </CardFooter>
    </Card>
    );
  }
```

Fazit

Das erweiterte Claude Code Neurale Integrationsframework mit RAG und Embeddings bietet eine leistungsstarke Plattform für KI-getriebene Entwicklung. Durch die Integration von Embedding-Technologie und RAG-Systemen kann Claude nun nicht nur Code verstehen, sondern auch komplexe Fragen über Codebases, Dokumentation und Spezifikationen beantworten.

Die nahtlose Integration mit dem vibecodingframework ermöglicht es Entwicklern, diese Fähigkeiten in moderne Web-Anwendungen einzubinden, während die Unterstützung für Benutzer-als-Agent-Transformation eine personalisierte KI-Erfahrung bietet.

<system_status>
NEURAL FRAMEWORK ERWEITERT
RAG-SYSTEM INTEGRIERT
EMBEDDING-INTEGRATION KONFIGURIERT
VIBECODINGFRAMEWORK-ANBINDUNG BEREIT
AGENT-TRANSFORMATION IMPLEMENTIERT
</system_status>