Claude-Flow v2.0.0-alpha.86 - Detaillierte Feature-Dokumentation

Inhaltsverzeichnis

Hauptdokumentation

Anhänge

Einführung

Claude-Flow v2.0.0-alpha.86 ist eine revolutionäre AI-Orchestrierungsplattform, die mehrere bahnbrechende Technologien kombiniert:

- 🐝 **Hive-Mind Intelligence**: Biologisch inspirierte Schwarm-Koordination
- 🧠 **Neural Networks**: 27+ kognitive Modelle mit Hardware-Beschleunigung
- 🔧 **87 MCP Tools**: Umfassendstes AI-Tool-Ökosystem
- 💾 **SQLite Memory**: Persistente, verteilte Wissensspeicherung
- 🔄 **DAA**: Selbstorganisierende, fehlertolerante Agenten
- 🪝 **Hooks**: Automatisierte Workflow-Integration
- 📊 **GitHub**: Nahtlose Repository-Verwaltung

Diese Dokumentation erklärt jedes Feature im Detail mit praktischen Beispielen.

Hive-Mind Intelligence

Konzept und Architektur

Die Hive-Mind Intelligence ist von biologischen Schwarm-Systemen inspiriert, insbesondere von Bienenvölkern. Das System nutzt eine hierarchische Struktur mit spezialisierten Agenten, die durch gemeinsamen Speicher und neuronale Mustererkennung koordiniert werden.

Hierarchie-Struktur

```
    👑 Queen Agent (Zentrale Koordination)
         │
  ┌──────┼────────┬─────────┬─────────┐
  │    │    │    │    │
🏗️ Architect 💻 Coder 🧪 Tester 📊 Analyst 🔍 Researcher
  │    │    │    │    │
Workers Workers Workers Workers Workers
```

Queen Agent (Koordinator)

Die Queen ist das zentrale Nervensystem des Hive-Mind:

```bash
# Queen spawnen mit spezifischen Fähigkeiten
npx claude-flow@alpha hive-mind spawn \
  --queen-type strategic \
  --consensus-algorithm majority \
  --delegation-strategy adaptive
```

Verantwortlichkeiten:

- **Task-Orchestrierung**: Verteilt Aufgaben basierend auf Agent-Fähigkeiten
- **Ressourcen-Management**: Überwacht CPU, Memory, und Netzwerk-Auslastung
- **Konsens-Koordination**: Sammelt Votes von Worker-Agenten für kritische Entscheidungen
- **Performance-Monitoring**: Trackt Success-Rates und optimiert Agent-Zuweisung
- **Konflikt-Resolution**: Löst Deadlocks und widersprüchliche Outputs

Worker-Agenten

Worker sind spezialisierte Agenten mit fokussierten Fähigkeiten:

```javascript
```

```javascript
// Beispiel: Worker-Konfiguration
const workerConfig = {
  researcher: {
    count: 2,
    capabilities: ["web-search", "documentation", "api-discovery"],
    memory: "512MB",
    priority: 8
  },
  coder: {
    count: 3,
    capabilities: ["implementation", "refactoring", "optimization"],
    memory: "1GB",
    priority: 9
  },
  tester: {
    count: 2,
    capabilities: ["unit-testing", "integration", "performance"],
    memory: "256MB",
    priority: 7
  }
}
```

Kollektive Intelligenz-Features

1. Consensus Voting

```bash
# Demokratische Entscheidungsfindung
npx claude-flow@alpha hive-mind consensus \
  --question "Should we use microservices architecture?" \
  --voters "architect,senior-dev,devops" \
  --algorithm weighted-majority
```

2. Memory Sharing

```bash
# Cross-Agent Memory Synchronisation
npx claude-flow@alpha memory share \
  --from researcher-alpha \
  --to "coder-beta,tester-gamma" \
  --namespace project-context
```

3. Neural Synchronization

```bash
# Neuronale Muster zwischen Agenten synchronisieren
npx claude-flow@alpha neural sync \
  --pattern successful-deployment \
  --broadcast-to all-agents
```

Praktisches Beispiel: E-Commerce Platform

```bash
#!/bin/bash
# hive-mind-ecommerce.sh

# 1. Hive initialisieren
npx claude-flow@alpha hive-mind init \
  --topology hierarchical \
  --agents 12 \
  --memory-size 2GB

# 2. Queen mit Projekt-Kontext spawnen
npx claude-flow@alpha hive-mind spawn \
  "Build complete e-commerce platform with microservices" \
  --queen-type strategic \
  --namespace ecommerce \
  --claude

# 3. Spezialisierte Worker hinzufügen
npx claude-flow@alpha agent spawn architect \
  --name "System Designer" \
  --capabilities "microservices,kubernetes,api-design"

npx claude-flow@alpha agent spawn coder \
  --name "Backend Developer" \
  --capabilities "nodejs,postgresql,redis" \
  --count 3

npx claude-flow@alpha agent spawn coder \
  --name "Frontend Developer" \
  --capabilities "react,typescript,tailwind" \
  --count 2

# 4. Live-Monitoring
npx claude-flow@alpha hive-mind monitor \
  --dashboard \
  --metrics "task-completion,agent-efficiency,memory-usage"
```

Neural Networks und Kognitive Modelle

WASM SIMD Acceleration

Claude-Flow nutzt WebAssembly (WASM) mit SIMD (Single Instruction, Multiple Data) für Hardware-beschleunigte neuronale Berechnungen:

```javascript
// WASM SIMD Konfiguration
const neuralConfig = {
  simd: {
    enabled: true,
    instructionSet: "wasm-simd128",
    parallelism: 4,
    memoryLimit: "512MB"
  },
  optimization: {
    quantization: "int8",    // Reduzierte Präzision für Geschwindigkeit
    pruning: 0.1,           // 10% Netzwerk-Pruning
    caching: true           // Aktivierung zwischen Layern cachen
  }
}
```

Kognitive Modell-Kategorien

Die 27+ kognitiven Modelle sind in 6 Hauptkategorien organisiert:

1. Koordinations-Modelle (5 Modelle)

- **Task Orchestrator**: Optimale Task-Verteilung
- **Resource Allocator**: Effiziente Ressourcen-Zuweisung
- **Conflict Resolver**: Deadlock-Auflösung
- **Priority Manager**: Dynamische Prioritäts-Anpassung
- **Load Balancer**: Agent-Workload-Verteilung

2. Lern-Modelle (6 Modelle)

- **Pattern Recognizer**: Erkennt wiederkehrende Entwicklungsmuster
- **Success Predictor**: Vorhersage von Task-Erfolgsraten
- **Error Analyzer**: Fehlerursachen-Analyse
- **Optimization Learner**: Lernt Performance-Optimierungen
- **Meta-Learner**: Lernt über Lernprozesse

- **Domain Adapter**: Anpassung an neue Domänen

3. Entscheidungs-Modelle (5 Modelle)

- **Decision Tree Navigator**: Komplexe Entscheidungsbäume
- **Risk Assessor**: Risikobewertung von Implementierungen
- **Trade-off Analyzer**: Kosten-Nutzen-Analyse
- **Strategy Selector**: Auswahl optimaler Strategien
- **Consensus Builder**: Konsens-Findung bei Konflikten

4. Kommunikations-Modelle (4 Modelle)

- **Intent Parser**: Verstehen von User-Intent
- **Context Maintainer**: Kontext-Kontinuität über Sessions
- **Response Generator**: Natürlichsprachliche Antworten
- **Translation Bridge**: Cross-Agent-Kommunikation

5. Performance-Modelle (4 Modelle)

- **Bottleneck Detector**: Identifikation von Performance-Engpässen
- **Optimization Suggester**: Vorschläge für Code-Optimierung
- **Cache Manager**: Intelligentes Caching
- **Resource Predictor**: Vorhersage von Ressourcen-Bedarf

6. Spezialisierte Modelle (3+ Modelle)

- **Code Quality Assessor**: Bewertung von Code-Qualität
- **Security Analyzer**: Sicherheitslücken-Erkennung
- **Documentation Generator**: Automatische Dokumentation

Training und Anwendung

bash

```bash
```

```
# Modell trainieren
npx claude-flow@alpha neural train \
  --model task-orchestrator \
  --data successful-workflows.json \
  --epochs 50 \
  --learning-rate 0.001 \
  --batch-size 32

# Vorhersage mit trainiertem Modell
npx claude-flow@alpha neural predict \
  --model task-orchestrator \
  --input current-state.json \
  --output-format detailed

# Performance analysieren
npx claude-flow@alpha cognitive analyze \
  --behavior "development-patterns" \
  --timeframe "last-7-days" \
  --export-insights patterns.json
```

Real-World Performance Metrics

```javascript
// Gemessene Performance-Verbesserungen durch Neural Networks
const performanceGains = {
  taskCompletion: {
    baseline: "100%",
    withNeuralNetworks: "284%",  // 2.84x schneller
    improvement: "+184%"
  },
  errorReduction: {
    baseline: "100 errors/1000 tasks",
    withNeuralNetworks: "15 errors/1000 tasks",
    improvement: "-85%"
  },
  resourceEfficiency: {
    baseline: "100% CPU/Memory",
    withNeuralNetworks: "67% CPU/Memory",
    improvement: "-33% Ressourcenverbrauch"
  }
}
```

MCP Tools System

Was ist das Model Context Protocol (MCP)?

MCP ist ein offener Standard von Anthropic, der eine einheitliche Schnittstelle zwischen AI-Modellen und externen Tools/Datenquellen bietet. Statt für jede Kombination aus AI-Modell und Tool eine eigene Integration zu bauen, bietet MCP eine standardisierte Kommunikationsschicht.

Die 87 MCP Tools in Claude-Flow

Claude-Flow implementiert 87 spezialisierte MCP Tools, organisiert in 10 Kategorien:

Tool-Kategorien Übersicht

1. **Swarm Management (12 Tools)**: Hive-Mind-Koordination
2. **Agent Control (15 Tools)**: Agent-Lifecycle-Management
3. **Task Orchestration (8 Tools)**: Workflow-Automation
4. **Memory Operations (10 Tools)**: Persistente Datenspeicherung
5. **Neural Processing (8 Tools)**: KI-Modell-Integration
6. **GitHub Integration (6 Tools)**: Repository-Management
7. **Performance Monitoring (7 Tools)**: System-Überwachung
8. **Workflow Automation (9 Tools)**: Pipeline-Orchestrierung
9. **Security & Compliance (6 Tools)**: Sicherheits-Features
10. **Utility Tools (6 Tools)**: Hilfsfunktionen

Praktische Anwendung

javascript

```javascript
// MCP Tool Integration Beispiel
async function orchestrateProject() {
  // 1. Swarm initialisieren
  await mcp__claude_flow__swarm_init({
    topology: "hierarchical",
    strategy: "auto",
    maxAgents: 10
  });

  // 2. Agenten spawnen
  const agents = await Promise.all([
    mcp__claude_flow__agent_spawn({
      type: "coordinator",
      name: "ProjectLead"
    }),
    mcp__claude_flow__agent_spawn({
      type: "coder",
      name: "BackendDev",
      capabilities: ["python", "fastapi"]
    }),
    mcp__claude_flow__agent_spawn({
      type: "tester",
      name: "QAEngineer"
    })
  ]);

  // 3. Task orchestrieren
  await mcp__claude_flow__task_orchestrate({
    task: "Build REST API with authentication",
    strategy: "parallel",
    priority: "high",
    agents: agents.map(a => a.id)
  });

  // 4. Memory speichern
  await mcp__claude_flow__memory_store({
    key: "project-status",
    value: { phase: "development", progress: 0.25 },
    namespace: "api-project"
  });

  // 5. Performance monitoren
  const metrics = await mcp__claude_flow__performance_monitor({
    metrics: ["task-completion", "agent-efficiency"],
    interval: 5000
  });
```

```
  return metrics;
}
```

Tool-Verkettung (Chaining)

```bash
# Komplexe Tool-Kette für Feature-Entwicklung
npx claude-flow@alpha chain execute \
  --tools "swarm_init → agent_spawn(5) → task_distribute → \
          parallel_execute → test_run → performance_analyze → \
          memory_store → report_generate" \
  --input "Build user authentication feature" \
  --output-format stream-json
```

SQLite Memory System

Architektur und Design

Das SQLite Memory System ist das persistente Gedächtnis von Claude-Flow, implementiert als relationale Datenbank mit 12 spezialisierten Tabellen:

```
.swarm/
└── memory.db (SQLite Database)
    ├── Core Tables (4)
    ├── Agent Tables (3)
    ├── Workflow Tables (2)
    └── Learning Tables (3)
```

Die 12 Tabellen im Detail

1. memory_store (Allgemeiner Key-Value-Speicher)

```sql
```

```
CREATE TABLE memory_store (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    key TEXT UNIQUE NOT NULL,
    value TEXT NOT NULL,
    namespace TEXT DEFAULT 'default',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ttl INTEGER DEFAULT NULL,
    compression TEXT DEFAULT NULL,
    INDEX idx_namespace (namespace),
    INDEX idx_created (created_at)
);
```

Verwendung: Speichern beliebiger Projekt-Daten, Konfigurationen, Zwischenergebnisse

2. sessions (Session-Management)

```sql
CREATE TABLE sessions (
    session_id TEXT PRIMARY KEY,
    user_id TEXT,
    project_name TEXT,
    context JSON,
    created_at TIMESTAMP,
    last_activity TIMESTAMP,
    status TEXT CHECK(status IN ('active', 'paused', 'completed')),
    metadata JSON
);
```

Verwendung: Verwalten von Entwicklungs-Sessions über mehrere Claude-Interaktionen

3. agents (Agent-Registry)

```sql
```

```sql
CREATE TABLE agents (
    agent_id TEXT PRIMARY KEY,
    type TEXT NOT NULL,
    name TEXT NOT NULL,
    status TEXT DEFAULT 'idle',
    capabilities JSON,
    memory_allocation INTEGER,
    created_at TIMESTAMP,
    last_task_at TIMESTAMP,
    performance_metrics JSON
);
```

Verwendung: Registrierung und Status-Tracking aller aktiven Agenten

4. tasks (Task-Tracking)

sql

```sql
CREATE TABLE tasks (
    task_id TEXT PRIMARY KEY,
    description TEXT,
    status TEXT DEFAULT 'pending',
    assigned_agent TEXT REFERENCES agents(agent_id),
    priority INTEGER DEFAULT 5,
    created_at TIMESTAMP,
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    result JSON,
    error_log TEXT
);
```

Verwendung: Verfolgung aller Tasks mit Status, Zuweisung und Ergebnissen

5. agent_memory (Agent-spezifischer Speicher)

sql

```sql
CREATE TABLE agent_memory (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    agent_id TEXT REFERENCES agents(agent_id),
    memory_key TEXT,
    memory_value TEXT,
    memory_type TEXT,
    importance REAL DEFAULT 0.5,
    access_count INTEGER DEFAULT 0,
    last_accessed TIMESTAMP,
    UNIQUE(agent_id, memory_key)
);
```

Verwendung: Individuelles Gedächtnis für jeden Agenten

6. shared_state (Cross-Agent geteilter Zustand)

```sql
CREATE TABLE shared_state (
    state_key TEXT PRIMARY KEY,
    state_value JSON,
    owner_agent TEXT,
    readers JSON,  -- Array of agent_ids
    writers JSON,  -- Array of agent_ids
    version INTEGER DEFAULT 1,
    locked BOOLEAN DEFAULT FALSE,
    lock_holder TEXT,
    updated_at TIMESTAMP
);
```

Verwendung: Gemeinsame Daten zwischen Agenten mit Zugriffskontrolle

7. events (Event-Log)

```sql
```

```sql
CREATE TABLE events (
    event_id INTEGER PRIMARY KEY AUTOINCREMENT,
    event_type TEXT NOT NULL,
    source TEXT NOT NULL,
    target TEXT,
    payload JSON,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed BOOLEAN DEFAULT FALSE,
    INDEX idx_timestamp (timestamp),
    INDEX idx_type (event_type)
);
```

Verwendung: Audit-Trail aller System-Events

8. patterns (Gelernte Muster)

```sql
CREATE TABLE patterns (
    pattern_id TEXT PRIMARY KEY,
    pattern_type TEXT,
    pattern_data JSON,
    confidence REAL,
    usage_count INTEGER DEFAULT 0,
    success_rate REAL,
    learned_at TIMESTAMP,
    last_used TIMESTAMP,
    metadata JSON
);
```

Verwendung: Speicherung erfolgreicher Entwicklungsmuster

9. performance_metrics (Performance-Tracking)

```sql
CREATE TABLE performance_metrics (
    metric_id INTEGER PRIMARY KEY AUTOINCREMENT,
    metric_name TEXT NOT NULL,
    metric_value REAL,
    unit TEXT,
    source TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    tags JSON,
    INDEX idx_name_time (metric_name, timestamp)
);
```

Verwendung: Detaillierte Performance-Metriken über Zeit

10. workflow_state (Workflow-Persistenz)

```sql
CREATE TABLE workflow_state (
    workflow_id TEXT PRIMARY KEY,
    workflow_name TEXT,
    current_stage TEXT,
    stages_completed JSON,
    stages_remaining JSON,
    context JSON,
    status TEXT,
    checkpoints JSON,
    created_at TIMESTAMP,
    updated_at TIMESTAMP
);
```

Verwendung: Speichern von Workflow-Zuständen für Wiederaufnahme

11. swarm_topology (Netzwerk-Topologie)

```sql
CREATE TABLE swarm_topology (
    node_id TEXT PRIMARY KEY,
    node_type TEXT,
    parent_id TEXT,
    children JSON,
    connections JSON,
    position JSON,  -- {x, y, z} für Visualisierung
    metadata JSON,
    active BOOLEAN DEFAULT TRUE
);
```

Verwendung: Abbildung der Hive-Mind-Struktur

12. consensus_state (Verteilter Konsens)

```sql
```

```sql
CREATE TABLE consensus_state (
    consensus_id TEXT PRIMARY KEY,
    question TEXT,
    votes JSON,
    result TEXT,
    algorithm TEXT,
    participants JSON,
    created_at TIMESTAMP,
    resolved_at TIMESTAMP,
    metadata JSON
);
```

Verwendung: Speicherung von Konsens-Entscheidungen

Memory Management Best Practices

```javascript
```

```javascript
// Effizientes Memory Management
class MemoryManager {
  constructor() {
    this.namespaces = new Map();
    this.compressionThreshold = 1024 * 10; // 10KB
  }

  async store(key, value, options = {}) {
    const {
      namespace = 'default',
      ttl = null,
      compress = 'auto',
      importance = 0.5
    } = options;

    // Auto-Kompression für große Daten
    let finalValue = value;
    let compression = null;

    if (compress === 'auto' && JSON.stringify(value).length > this.compressionThreshold) {
      finalValue = await this.compress(value);
      compression = 'gzip';
    }

    // In SQLite speichern
    await db.run(`
      INSERT OR REPLACE INTO memory_store
      (key, value, namespace, ttl, compression)
      VALUES (?, ?, ?, ?, ?)
    `, [key, finalValue, namespace, ttl, compression]);

    // Cache invalidieren
    this.invalidateCache(namespace, key);
  }

  async query(pattern, options = {}) {
    const {
      namespace = null,
      limit = 100,
      orderBy = 'created_at DESC',
      since = null
    } = options;

    let query = 'SELECT * FROM memory_store WHERE key LIKE ?';
    const params = [`%${pattern}%`];
```

```
    if (namespace) {
      query += ' AND namespace = ?';
      params.push(namespace);
    }

    if (since) {
      query += ' AND created_at > ?';
      params.push(since);
    }

    query += ` ORDER BY ${orderBy} LIMIT ?`;
    params.push(limit);

    return await db.all(query, params);
  }

  async cleanup() {
    // TTL-abgelaufene Einträge löschen
    await db.run(`
      DELETE FROM memory_store
      WHERE ttl IS NOT NULL
      AND datetime('now') > datetime(created_at, '+' || ttl || ' seconds')
    `);

    // Vacuum für Optimierung
    await db.run('VACUUM');
  }
}
```

Dynamic Agent Architecture (DAA)

Konzept

Die Dynamic Agent Architecture ermöglicht selbstorganisierende, adaptive Agenten-Netzwerke:

```javascript
```

```javascript
// DAA Konfiguration
const daaConfig = {
  autonomy: {
    selfOrganization: true,     // Agenten organisieren sich selbst
    autoScaling: true,          // Automatisches Hoch-/Runterskalieren
    faultTolerance: true,       // Automatische Fehlerbehandlung
    loadBalancing: true         // Dynamische Last-Verteilung
  },
  adaptation: {
    learningRate: 0.1,          // Wie schnell Agenten lernen
    evolutionCycles: 100,       // Iterations für Evolution
    mutationRate: 0.05,         // Variations-Rate
    crossoverRate: 0.7          // Knowledge-Sharing-Rate
  },
  resilience: {
    redundancy: 2,              // Backup-Agenten pro kritischer Rolle
    heartbeatInterval: 5000,    // Health-Check-Intervall (ms)
    recoveryTimeout: 30000,     // Max Recovery-Zeit (ms)
    circuitBreaker: {
      threshold: 5,             // Fehler vor Circuit-Break
      timeout: 60000            // Circuit-Breaker-Timeout
    }
  }
}
```

Selbstorganisation

```bash
# DAA mit Selbstorganisation starten
npx claude-flow@alpha daa enable \
  --mode autonomous \
  --min-agents 3 \
  --max-agents 20 \
  --optimization-goal "minimize-time"

# Agenten passen sich automatisch an:
# - Spawnen neue Agenten bei hoher Last
# - Terminieren idle Agenten
# - Re-balancieren Tasks zwischen Agenten
# - Lernen optimale Konfigurationen
```

Fehlertoleranz

```javascript
```

```
// Automatische Fehlerbehandlung
class ResilientAgent {
  constructor(config) {
    this.retryPolicy = {
      maxAttempts: 3,
      backoff: 'exponential',
      initialDelay: 1000
    };

    this.circuitBreaker = new CircuitBreaker({
      threshold: 5,
      timeout: 60000,
      onOpen: () => this.handleCircuitOpen(),
      onClose: () => this.handleCircuitClose()
    });
  }

  async executeTask(task) {
    return this.circuitBreaker.fire(async () => {
      return await this.retryWithBackoff(async () => {
        // Wenn Agent ausfällt, übernimmt automatisch ein anderer
        try {
          return await this.performTask(task);
        } catch (error) {
          // Automatisches Failover zu Backup-Agent
          return await this.failoverToBackup(task);
        }
      });
    });
  }
}
```

Advanced Hooks System

Hook-Typen und Lifecycle

```javascript
```

```
// Vollständiger Hook-Lifecycle
const hookLifecycle = {
  // Globale Hooks
  global: {
    preInit: "./hooks/pre-init.sh",          // Vor System-Init
    postInit: "./hooks/post-init.sh",        // Nach System-Init
    preShutdown: "./hooks/pre-shutdown.sh",  // Vor Shutdown
    postShutdown: "./hooks/post-shutdown.sh" // Nach Shutdown
  },

  // Tool-Hooks
  tools: {
    preToolUse: "./hooks/pre-tool.js",       // Vor Tool-Ausführung
    postToolUse: "./hooks/post-tool.js",     // Nach Tool-Ausführung
    onToolError: "./hooks/tool-error.js"     // Bei Tool-Fehler
  },

  // Agent-Hooks
  agents: {
    onSpawn: "./hooks/agent-spawn.js",       // Bei Agent-Erstellung
    onTerminate: "./hooks/agent-terminate.js", // Bei Agent-Beendigung
    onIdle: "./hooks/agent-idle.js",         // Wenn Agent idle
    onBusy: "./hooks/agent-busy.js"          // Wenn Agent beschäftigt
  },

  // Workflow-Hooks
  workflow: {
    onStageComplete: "./hooks/stage-complete.js",
    onCheckpoint: "./hooks/checkpoint.js",
    onRollback: "./hooks/rollback.js"
  }
}
```

Praktisches Hook-Beispiel

javascript

```
// hooks/pre-tool.js - Automatische Code-Formatierung
module.exports = async function preToolHook(context) {
  const { tool, args, agent } = context;

  // Bei File-Writes automatisch formatieren
  if (tool === 'Write' || tool === 'Edit') {
    const filePath = args.path;
    const content = args.content;

    // Sprach-spezifische Formatierung
    if (filePath.endsWith('.py')) {
      args.content = await formatPython(content);
      log(`Formatted Python file: ${filePath}`);
    } else if (filePath.endsWith('.js') || filePath.endsWith('.ts')) {
      args.content = await formatJavaScript(content);
      log(`Formatted JavaScript file: ${filePath}`);
    }

    // Linting
    const lintErrors = await lint(filePath, args.content);
    if (lintErrors.length > 0) {
      // Hook kann Tool-Ausführung verhindern
      return {
        allow: false,
        reason: `Linting errors found: ${lintErrors.join(', ')}`
      };
    }
  }

  return { allow: true };
}
```

Hook-Konfiguration in settings.json

```
json
```

```json
{
  "hooks": [
    {
      "matcher": "Edit|Write|MultiEdit",
      "hooks": [
        {
          "type": "command",
          "command": "prettier --write \"$CLAUDE_FILE_PATHS\""
        },
        {
          "type": "command",
          "command": "eslint --fix \"$CLAUDE_FILE_PATHS\""
        }
      ]
    },
    {
      "matcher": "Bash",
      "hooks": [
        {
          "type": "script",
          "script": "./hooks/validate-command.js"
        }
      ]
    },
    {
      "matcher": ".*Test.*",
      "hooks": [
        {
          "type": "command",
          "command": "npm test -- --coverage"
        }
      ]
    }
  ]
}
```

GitHub Integration

Die 6 Spezialisierten GitHub-Modi

Claude-Flow bietet 6 spezialisierte Modi für umfassende GitHub-Integration:

1. GH-Coordinator (Repository-Koordination)

bash

```
npx claude-flow@alpha github gh-coordinator \
  --analysis-type comprehensive \
  --target ./src \
  --output report.md

# Funktionen:
# - Code-Analyse über mehrere Repositories
# - Dependency-Tracking
# - Cross-Repo-Refactoring
# - Team-Koordination
```

2. PR-Manager (Pull Request Management)

```bash
npx claude-flow@alpha github pr-manager \
  --action review \
  --multi-reviewer \
  --ai-powered \
  --auto-merge-on-approval

# Funktionen:
# - Automatische PR-Reviews
# - Multi-Agenten-Review-Teams
# - Konflikt-Resolution
# - Auto-Merge-Strategien
```

3. Issue-Tracker (Issue-Verwaltung)

```bash
npx claude-flow@alpha github issue-tracker \
  --mode manage \
  --auto-label \
  --auto-assign \
  --project-coordination

# Funktionen:
# - Automatische Issue-Klassifizierung
# - Intelligente Agent-Zuweisung
# - Sprint-Planning-Integration
# - Bug-Priorisierung
```

4. Release-Manager (Release-Koordination)

```bash
npx claude-flow@alpha github release-manager \
  --version 2.0.0 \
  --auto-changelog \
  --semantic-versioning \
  --deployment-coordination

# Funktionen:
# - Automatische Changelog-Generierung
# - Semantic Versioning
# - Multi-Repo-Releases
# - Deployment-Orchestrierung
```

5. Repo-Architect (Repository-Struktur-Optimierung)

```bash
npx claude-flow@alpha github repo-architect \
  --optimize \
  --structure-analysis \
  --refactor-suggestions \
  --monorepo-migration

# Funktionen:
# - Repository-Struktur-Analyse
# - Refactoring-Vorschläge
# - Monorepo-Migration
# - Dependency-Optimierung
```

6. Sync-Coordinator (Multi-Repo-Synchronisation)

```bash
npx claude-flow@alpha github sync-coordinator \
  --repos "frontend,backend,shared-libs" \
  --sync-strategy selective \
  --conflict-resolution automatic

# Funktionen:
# - Cross-Repo-Synchronisation
# - Shared-Code-Management
# - Version-Alignment
# - Konflikt-Auflösung
```

GitHub Workflow Integration

```yaml
# .github/workflows/claude-flow.yml
name: Claude-Flow AI Development

on:
  issues:
    types: [opened, labeled]
  pull_request:
    types: [opened, synchronize]

jobs:
  ai-development:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Claude-Flow
        run: |
          npm install -g claude-flow@alpha
          claude-flow init --force

      - name: Auto-Assign Issue to AI Agent
        if: github.event_name == 'issues'
        run: |
          claude-flow github issue-tracker \
            --issue "${{ github.event.issue.number }}" \
            --auto-assign \
            --spawn-agent

      - name: AI Code Review
        if: github.event_name == 'pull_request'
        run: |
          claude-flow github pr-manager \
            --pr "${{ github.event.pull_request.number }}" \
            --review \
            --suggest-improvements
```

Anhang A: Vollständige Liste der 87 MCP Tools

Swarm Management Tools (12)

1. **swarm_init** - Swarm initialisieren mit Topologie
2. **swarm_status** - Swarm-Status abfragen
3. **swarm_think** - Kollektive Problemlösung

4. **swarm_terminate** - Swarm beenden
5. **swarm_pause** - Swarm pausieren
6. **swarm_resume** - Swarm fortsetzen
7. **swarm_scale** - Swarm skalieren
8. **swarm_rebalance** - Load-Balancing
9. **swarm_checkpoint** - Checkpoint erstellen
10. **swarm_rollback** - Zu Checkpoint zurück
11. **swarm_merge** - Swarms zusammenführen
12. **swarm_split** - Swarm aufteilen

Agent Control Tools (15)

13. **agent_spawn** - Agent erstellen
14. **agent_terminate** - Agent beenden
15. **agent_status** - Agent-Status
16. **agent_assign** - Task zuweisen
17. **agent_reassign** - Task neu zuweisen
18. **agent_pause** - Agent pausieren
19. **agent_resume** - Agent fortsetzen
20. **agent_upgrade** - Agent-Fähigkeiten erweitern
21. **agent_downgrade** - Fähigkeiten reduzieren
22. **agent_clone** - Agent klonen
23. **agent_migrate** - Agent migrieren
24. **agent_communicate** - Inter-Agent-Kommunikation
25. **agent_synchronize** - Agenten synchronisieren
26. **agent_benchmark** - Performance testen
27. **agent_profile** - Profiling

Task Orchestration Tools (8)

28. **task_create** - Task erstellen
29. **task_orchestrate** - Task orchestrieren
30. **task_distribute** - Tasks verteilen
31. **task_prioritize** - Prioritäten setzen
32. **task_dependencies** - Abhängigkeiten verwalten
33. **task_monitor** - Task-Monitoring

34. **task_retry** - Task wiederholen
35. **task_cancel** - Task abbrechen

Memory Operations Tools (10)

36. **memory_store** - Daten speichern
37. **memory_retrieve** - Daten abrufen
38. **memory_query** - Daten suchen
39. **memory_update** - Daten aktualisieren
40. **memory_delete** - Daten löschen
41. **memory_share** - Memory teilen
42. **memory_sync** - Memory synchronisieren
43. **memory_compress** - Memory komprimieren
44. **memory_export** - Memory exportieren
45. **memory_import** - Memory importieren

Neural Processing Tools (8)

46. **neural_train** - Modell trainieren
47. **neural_predict** - Vorhersage
48. **neural_sync** - Neuronale Synchronisation
49. **neural_evolve** - Modell-Evolution
50. **neural_prune** - Modell-Pruning
51. **neural_quantize** - Quantisierung
52. **neural_ensemble** - Ensemble-Learning
53. **neural_transfer** - Transfer-Learning

GitHub Integration Tools (6)

54. **github_coordinator** - Repo-Koordination
55. **github_pr_manager** - PR-Management
56. **github_issue_tracker** - Issue-Tracking
57. **github_release_manager** - Release-Management
58. **github_repo_architect** - Repo-Optimierung
59. **github_sync_coordinator** - Multi-Repo-Sync

Performance Monitoring Tools (7)

60. **performance_monitor** - Performance überwachen

61. **performance_analyze** - Performance analysieren
62. **performance_optimize** - Performance optimieren
63. **bottleneck_detect** - Engpässe finden
64. **resource_monitor** - Ressourcen überwachen
65. **metric_collect** - Metriken sammeln
66. **alert_manage** - Alerts verwalten

Workflow Automation Tools (9)

67. **workflow_create** - Workflow erstellen
68. **workflow_execute** - Workflow ausführen
69. **workflow_schedule** - Workflow planen
70. **pipeline_create** - Pipeline erstellen
71. **pipeline_execute** - Pipeline ausführen
72. **batch_process** - Batch-Verarbeitung
73. **parallel_execute** - Parallele Ausführung
74. **chain_execute** - Tool-Verkettung
75. **conditional_execute** - Bedingte Ausführung

Security & Compliance Tools (6)

76. **security_scan** - Sicherheitsscan
77. **vulnerability_check** - Schwachstellen prüfen
78. **compliance_audit** - Compliance-Audit
79. **access_control** - Zugriffskontrolle
80. **encryption_manage** - Verschlüsselung
81. **secret_manage** - Secrets verwalten

Utility Tools (6)

82. **consensus_vote** - Konsens-Abstimmung
83. **health_check** - System-Health-Check
84. **diagnostic_run** - Diagnose ausführen
85. **backup_create** - Backup erstellen
86. **restore_backup** - Backup wiederherstellen
87. **system_reset** - System zurücksetzen

Anhang B: Die 27+ Kognitiven Modelle

Vollständige Liste (33 Modelle)

Koordinations-Modelle

1. **task-orchestrator** - Optimale Task-Verteilung
2. **resource-allocator** - Ressourcen-Management
3. **conflict-resolver** - Konflikt-Auflösung
4. **priority-manager** - Prioritäts-Verwaltung
5. **load-balancer** - Last-Verteilung

Lern-Modelle

6. **pattern-recognizer** - Muster-Erkennung
7. **success-predictor** - Erfolgs-Vorhersage
8. **error-analyzer** - Fehler-Analyse
9. **optimization-learner** - Optimierungs-Lernen
10. **meta-learner** - Meta-Learning
11. **domain-adapter** - Domänen-Anpassung

Entscheidungs-Modelle

12. **decision-tree** - Entscheidungsbaum-Navigation
13. **risk-assessor** - Risiko-Bewertung
14. **trade-off-analyzer** - Trade-off-Analyse
15. **strategy-selector** - Strategie-Auswahl
16. **consensus-builder** - Konsens-Bildung

Kommunikations-Modelle

17. **intent-parser** - Intent-Erkennung
18. **context-maintainer** - Kontext-Erhaltung
19. **response-generator** - Antwort-Generierung
20. **translation-bridge** - Übersetzungs-Brücke

Performance-Modelle

21. **bottleneck-detector** - Engpass-Erkennung
22. **optimization-suggester** - Optimierungs-Vorschläge
23. **cache-manager** - Cache-Verwaltung

24. **resource-predictor** - Ressourcen-Vorhersage

Code-Qualitäts-Modelle

25. **quality-assessor** - Qualitäts-Bewertung
26. **complexity-analyzer** - Komplexitäts-Analyse
27. **refactor-suggester** - Refactoring-Vorschläge

Sicherheits-Modelle

28. **security-analyzer** - Sicherheits-Analyse
29. **vulnerability-detector** - Schwachstellen-Erkennung
30. **threat-assessor** - Bedrohungs-Bewertung

Dokumentations-Modelle

31. **doc-generator** - Dokumentations-Generierung
32. **api-documenter** - API-Dokumentation
33. **comment-generator** - Kommentar-Generierung

Anhang C: SQLite Tabellen-Schema

Vollständiges Schema mit Indizes und Constraints

sql

```sql
-- 1. Memory Store
CREATE TABLE memory_store (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    key TEXT UNIQUE NOT NULL,
    value TEXT NOT NULL,
    namespace TEXT DEFAULT 'default',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ttl INTEGER DEFAULT NULL,
    compression TEXT DEFAULT NULL,
    importance REAL DEFAULT 0.5,
    access_count INTEGER DEFAULT 0
);
CREATE INDEX idx_memory_namespace ON memory_store(namespace);
CREATE INDEX idx_memory_created ON memory_store(created_at);
CREATE INDEX idx_memory_importance ON memory_store(importance DESC);

-- 2. Sessions
CREATE TABLE sessions (
    session_id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL,
    project_name TEXT,
    context JSON,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_activity TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status TEXT CHECK(status IN ('active', 'paused', 'completed', 'failed')),
    metadata JSON,
    parent_session TEXT REFERENCES sessions(session_id)
);
CREATE INDEX idx_sessions_user ON sessions(user_id);
CREATE INDEX idx_sessions_status ON sessions(status);
CREATE INDEX idx_sessions_activity ON sessions(last_activity DESC);

-- 3. Agents
CREATE TABLE agents (
    agent_id TEXT PRIMARY KEY,
    type TEXT NOT NULL CHECK(type IN ('coordinator', 'coder', 'tester', 'architect',
                                      'researcher', 'analyst', 'specialist')),
    name TEXT NOT NULL,
    status TEXT DEFAULT 'idle' CHECK(status IN ('idle', 'busy', 'paused', 'terminated')),
    capabilities JSON,
    memory_allocation INTEGER DEFAULT 256,
    priority INTEGER DEFAULT 5 CHECK(priority BETWEEN 1 AND 10),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_task_at TIMESTAMP,
    performance_metrics JSON,
```

```sql
    parent_agent TEXT REFERENCES agents(agent_id)
);
CREATE INDEX idx_agents_type ON agents(type);
CREATE INDEX idx_agents_status ON agents(status);
CREATE INDEX idx_agents_priority ON agents(priority DESC);

-- 4. Tasks
CREATE TABLE tasks (
    task_id TEXT PRIMARY KEY,
    description TEXT NOT NULL,
    status TEXT DEFAULT 'pending' CHECK(status IN ('pending', 'running', 'completed',
                                              'failed', 'cancelled')),
    assigned_agent TEXT REFERENCES agents(agent_id),
    priority INTEGER DEFAULT 5 CHECK(priority BETWEEN 1 AND 10),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    estimated_duration INTEGER,
    actual_duration INTEGER,
    result JSON,
    error_log TEXT,
    retry_count INTEGER DEFAULT 0,
    parent_task TEXT REFERENCES tasks(task_id)
);
CREATE INDEX idx_tasks_status ON tasks(status);
CREATE INDEX idx_tasks_agent ON tasks(assigned_agent);
CREATE INDEX idx_tasks_priority ON tasks(priority DESC, created_at ASC);

-- 5. Agent Memory
CREATE TABLE agent_memory (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    agent_id TEXT REFERENCES agents(agent_id) ON DELETE CASCADE,
    memory_key TEXT NOT NULL,
    memory_value TEXT NOT NULL,
    memory_type TEXT DEFAULT 'general',
    importance REAL DEFAULT 0.5 CHECK(importance BETWEEN 0 AND 1),
    access_count INTEGER DEFAULT 0,
    last_accessed TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ttl INTEGER DEFAULT NULL,
    UNIQUE(agent_id, memory_key)
);
CREATE INDEX idx_agent_memory_agent ON agent_memory(agent_id);
CREATE INDEX idx_agent_memory_type ON agent_memory(memory_type);
CREATE INDEX idx_agent_memory_importance ON agent_memory(importance DESC);

-- 6. Shared State
CREATE TABLE shared_state (
```

```sql
    state_key TEXT PRIMARY KEY,
    state_value JSON NOT NULL,
    owner_agent TEXT REFERENCES agents(agent_id),
    readers JSON DEFAULT '[]',
    writers JSON DEFAULT '[]',
    version INTEGER DEFAULT 1,
    locked BOOLEAN DEFAULT FALSE,
    lock_holder TEXT REFERENCES agents(agent_id),
    lock_acquired TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_shared_state_owner ON shared_state(owner_agent);
CREATE INDEX idx_shared_state_locked ON shared_state(locked);

-- 7. Events
CREATE TABLE events (
    event_id INTEGER PRIMARY KEY AUTOINCREMENT,
    event_type TEXT NOT NULL,
    source TEXT NOT NULL,
    target TEXT,
    payload JSON,
    severity TEXT DEFAULT 'info' CHECK(severity IN ('debug', 'info', 'warning', 'error', 'critical')),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed BOOLEAN DEFAULT FALSE,
    processing_result JSON
);
CREATE INDEX idx_events_timestamp ON events(timestamp DESC);
CREATE INDEX idx_events_type ON events(event_type);
CREATE INDEX idx_events_severity ON events(severity);
CREATE INDEX idx_events_processed ON events(processed);

-- 8. Patterns
CREATE TABLE patterns (
    pattern_id TEXT PRIMARY KEY,
    pattern_type TEXT NOT NULL,
    pattern_data JSON NOT NULL,
    confidence REAL DEFAULT 0.5 CHECK(confidence BETWEEN 0 AND 1),
    usage_count INTEGER DEFAULT 0,
    success_rate REAL DEFAULT 0 CHECK(success_rate BETWEEN 0 AND 1),
    failure_rate REAL DEFAULT 0 CHECK(failure_rate BETWEEN 0 AND 1),
    learned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_used TIMESTAMP,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    metadata JSON
);
CREATE INDEX idx_patterns_type ON patterns(pattern_type);
CREATE INDEX idx_patterns_confidence ON patterns(confidence DESC);
```

```sql
CREATE INDEX idx_patterns_success ON patterns(success_rate DESC);
CREATE INDEX idx_patterns_usage ON patterns(usage_count DESC);

-- 9. Performance Metrics
CREATE TABLE performance_metrics (
    metric_id INTEGER PRIMARY KEY AUTOINCREMENT,
    metric_name TEXT NOT NULL,
    metric_value REAL NOT NULL,
    unit TEXT,
    source TEXT NOT NULL,
    target TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    aggregation_period TEXT DEFAULT 'instant',
    tags JSON,
    metadata JSON
);
CREATE INDEX idx_metrics_name_time ON performance_metrics(metric_name, timestamp DESC);
CREATE INDEX idx_metrics_source ON performance_metrics(source);
CREATE INDEX idx_metrics_timestamp ON performance_metrics(timestamp DESC);

-- 10. Workflow State
CREATE TABLE workflow_state (
    workflow_id TEXT PRIMARY KEY,
    workflow_name TEXT NOT NULL,
    workflow_definition JSON,
    current_stage TEXT,
    stages_completed JSON DEFAULT '[]',
    stages_remaining JSON DEFAULT '[]',
    context JSON,
    variables JSON,
    status TEXT DEFAULT 'created' CHECK(status IN ('created', 'running', 'paused',
                                        'completed', 'failed', 'cancelled')),
    checkpoints JSON DEFAULT '[]',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    started_at TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP,
    error_log TEXT
);
CREATE INDEX idx_workflow_status ON workflow_state(status);
CREATE INDEX idx_workflow_updated ON workflow_state(updated_at DESC);

-- 11. Swarm Topology
CREATE TABLE swarm_topology (
    node_id TEXT PRIMARY KEY,
    node_type TEXT NOT NULL CHECK(node_type IN ('queen', 'lead', 'worker')),
    parent_id TEXT REFERENCES swarm_topology(node_id),
```

```sql
    children JSON DEFAULT '[]',
    connections JSON DEFAULT '[]',
    position JSON,
    capabilities JSON,
    load_factor REAL DEFAULT 0 CHECK(load_factor BETWEEN 0 AND 1),
    health_score REAL DEFAULT 1 CHECK(health_score BETWEEN 0 AND 1),
    metadata JSON,
    active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_heartbeat TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_topology_type ON swarm_topology(node_type);
CREATE INDEX idx_topology_parent ON swarm_topology(parent_id);
CREATE INDEX idx_topology_active ON swarm_topology(active);
CREATE INDEX idx_topology_health ON swarm_topology(health_score);

-- 12. Consensus State
CREATE TABLE consensus_state (
    consensus_id TEXT PRIMARY KEY,
    question TEXT NOT NULL,
    options JSON NOT NULL,
    votes JSON DEFAULT '{}',
    weights JSON DEFAULT '{}',
    result TEXT,
    confidence REAL,
    algorithm TEXT DEFAULT 'majority' CHECK(algorithm IN ('majority', 'weighted',
                                                'byzantine', 'raft')),
    participants JSON NOT NULL,
    required_votes INTEGER,
    received_votes INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    resolved_at TIMESTAMP,
    timeout_at TIMESTAMP,
    metadata JSON
);
CREATE INDEX idx_consensus_resolved ON consensus_state(resolved_at);
CREATE INDEX idx_consensus_timeout ON consensus_state(timeout_at);

-- Triggers für automatische Updates
CREATE TRIGGER update_memory_timestamp
AFTER UPDATE ON memory_store
BEGIN
  UPDATE memory_store SET updated_at = CURRENT_TIMESTAMP
  WHERE id = NEW.id;
END;

CREATE TRIGGER update_session_activity
```

```
AFTER UPDATE ON sessions
BEGIN
  UPDATE sessions SET last_activity = CURRENT_TIMESTAMP
  WHERE session_id = NEW.session_id;
END;

CREATE TRIGGER update_workflow_timestamp
AFTER UPDATE ON workflow_state
BEGIN
  UPDATE workflow_state SET updated_at = CURRENT_TIMESTAMP
  WHERE workflow_id = NEW.workflow_id;
END;

-- Views für häufige Abfragen
CREATE VIEW active_agents AS
SELECT * FROM agents
WHERE status IN ('idle', 'busy')
ORDER BY priority DESC;

CREATE VIEW pending_tasks AS
SELECT t.*, a.name as agent_name
FROM tasks t
LEFT JOIN agents a ON t.assigned_agent = a.agent_id
WHERE t.status = 'pending'
ORDER BY t.priority DESC, t.created_at ASC;

CREATE VIEW recent_patterns AS
SELECT * FROM patterns
WHERE last_used > datetime('now', '-7 days')
ORDER BY confidence DESC, success_rate DESC;
```

Anhang D: Agent-Typen und Spezialisierungen

Vollständige Agent-Typologie

1. Coordinator (Koordinatoren)

```javascript
```

```javascript
{
  type: "coordinator",
  subtypes: [
    "project-manager",     // Projekt-Koordination
    "team-lead",           // Team-Führung
    "scrum-master",        // Agile-Koordination
    "tech-lead"            // Technische Führung
  ],
  capabilities: [
    "task-distribution",
    "resource-allocation",
    "conflict-resolution",
    "progress-tracking",
    "team-coordination"
  ]
}
```

2. Architect (Architekten)

```javascript
{
  type: "architect",
  subtypes: [
    "system-architect",    // System-Design
    "solution-architect",  // Lösungs-Architektur
    "cloud-architect",     // Cloud-Infrastruktur
    "data-architect"       // Daten-Architektur
  ],
  capabilities: [
    "system-design",
    "pattern-selection",
    "technology-evaluation",
    "scalability-planning",
    "architecture-documentation"
  ]
}
```

3. Coder (Entwickler)

```javascript
```

```
{
  type: "coder",
  subtypes: [
    "backend-developer",   // Backend-Entwicklung
    "frontend-developer",  // Frontend-Entwicklung
    "fullstack-developer", // Full-Stack
    "mobile-developer",    // Mobile-Entwicklung
    "embedded-developer"   // Embedded-Systeme
  ],
  capabilities: [
    "implementation",
    "refactoring",
    "debugging",
    "optimization",
    "code-review"
  ],
  languages: [
    "python", "javascript", "typescript", "java",
    "go", "rust", "c++", "swift", "kotlin"
  ]
}
```

4. Tester (Qualitätssicherung)

```javascript
{
  type: "tester",
  subtypes: [
    "unit-tester",         // Unit-Tests
    "integration-tester",  // Integration-Tests
    "e2e-tester",          // End-to-End-Tests
    "performance-tester",  // Performance-Tests
    "security-tester"      // Security-Tests
  ],
  capabilities: [
    "test-creation",
    "test-execution",
    "bug-reporting",
    "coverage-analysis",
    "test-automation"
  ]
}
```

5. Researcher (Forscher)

```javascript
{
  type: "researcher",
  subtypes: [
    "technology-researcher",  // Technologie-Forschung
    "market-researcher",      // Markt-Analyse
    "user-researcher",        // User-Research
    "competitive-analyst"     // Wettbewerbs-Analyse
  ],
  capabilities: [
    "information-gathering",
    "analysis",
    "documentation",
    "trend-identification",
    "recommendation"
  ]
}
```

6. Analyst (Analysten)

```javascript
{
  type: "analyst",
  subtypes: [
    "business-analyst",      // Business-Analyse
    "data-analyst",          // Daten-Analyse
    "system-analyst",        // System-Analyse
    "performance-analyst"    // Performance-Analyse
  ],
  capabilities: [
    "data-analysis",
    "reporting",
    "visualization",
    "insight-generation",
    "requirement-analysis"
  ]
}
```

7. Specialist (Spezialisten)

```javascript
```

```
{
  type: "specialist",
  subtypes: [
    "devops-engineer",      // DevOps
    "ml-engineer",          // Machine Learning
    "security-engineer",    // Security
    "database-admin",       // Datenbank
    "network-engineer",     // Netzwerk
    "ui-ux-designer"        // UI/UX
  ],
  capabilities: [
    "specialized-implementation",
    "expert-consultation",
    "tool-integration",
    "best-practice-enforcement",
    "specialized-optimization"
  ]
}
```

Zusammenfassung

Claude-Flow v2.0.0-alpha.86 repräsentiert einen Paradigmenwechsel in der AI-gestützten Softwareentwicklung durch:

1. **Biologisch inspirierte Intelligenz**: Hive-Mind-Architektur mit Queen-geführter Koordination
2. **Hardware-beschleunigte KI**: 27+ kognitive Modelle mit WASM SIMD
3. **Umfassendes Tool-Ökosystem**: 87 spezialisierte MCP Tools
4. **Persistente Intelligenz**: SQLite-basiertes 12-Tabellen-Memory-System
5. **Selbstorganisation**: Dynamic Agent Architecture mit Fehlertoleranz
6. **Workflow-Automation**: Advanced Hooks für nahtlose Integration
7. **Repository-Intelligence**: 6 spezialisierte GitHub-Modi

Diese Technologien ermöglichen eine **2.8-4.4x Geschwindigkeitssteigerung** bei gleichzeitiger **85% Fehlerreduktion** und **33% geringerem Ressourcenverbrauch**.

Dokumentation erstellt für Claude-Flow v2.0.0-alpha.86 Stand: August 2025