ChatGPT

Strategie und Konfigurationsdateien für den autonomen Codex-Agent

Ausgangslage (Analyse): Das Repository **Agent-NN** stellt ein komplexes Multi-Agenten-System mit LLM-Integration dar. Die **Smolitux-UI** (Frontend) ist bereits umgesetzt, jedoch muss die Agent-NN-Plattform noch lokal installiert und konfiguriert werden. Zudem fehlen noch einige wichtige Agenten-Komponenten – explizit genannt werden ein **LOH-Agent**, ein **Agent-Setup** und ein **Agent-Framework** [1]. Des Weiteren sind bestimmte Kernfunktionen unvollständig: So ist bspw. die Implementierung des **SupervisorAgent** zwar begonnen, aber zugehörige Tests (`test_supervisor_agent.py`) fehlen noch, ebenso eine konsistente Logging- und Fehlerbehandlungs-Komponente (`logging_util.py`) [2]. Die Dokumentationsstruktur (Benutzerhandbuch, Architektur, API, etc.) ist angelegt, jedoch inhaltlich unvollständig.

Ziel: Codex (ChatGPT Codex im Browser) soll als autonomer Coding-Agent diese Lücken schließen und das MVP (Minimum Viable Product) in mehreren Phasen fertigstellen. Dabei soll der Agent selbstständig den Fortschritt überwachen und in die nächste Phase wechseln, sobald die vorherige abgeschlossen ist. Im Folgenden wird ein in **5 Phasen** gegliederter Strategieplan beschrieben – von Analyse über Planung und Umsetzung bis hin zu Qualitätssicherung und Dokumentations-Finalisierung – sowie die dafür erforderlichen Konfigurationsdateien bereitgestellt. Diese Dateien steuern das Verhalten des Codex-Agenten und definieren Rollen, Aufgaben und Richtlinien für die autonome Entwicklung.

Phasenplan (Strategie)

Phase 1: Analyse der Architektur

In dieser initialen Phase übernimmt Codex die Rolle eines **Architektur-Analysten**. Der Agent durchsucht das Repository, liest Quellcode und Dokumentation, um die bestehende **Architektur** vollständig zu verstehen. Dabei werden Schwachstellen oder Lücken dokumentiert. Wichtige Fragen in dieser Phase: *Welche Komponenten sind bereits implementiert? Wo gibt es Pseudocode oder Platzhalter? Welche Funktionen fehlen für ein MVP?* Der Agent soll insbesondere prüfen, ob die vorgesehenen Agenten (Supervisor, Worker, etc.) schon funktionsfähig sind und wo **Refactoring-Bedarf** besteht. Als Ergebnis der Analyse erzeugt Codex einen kurzen **Architekturbericht** (z.B. in `docs/architecture/overview.md`), der den aktuellen Stand und empfohlene Verbesserungen zusammenfasst. Wichtig ist auch die **Anforderungsanalyse**: der Agent validiert, ob die im Repository (README/Roadmap) definierten MVP-Ziele alle abgedeckt sind, und listet fehlende Features auf (z.B. **LOH-Agent**, Setup-Agent etc. aus der README [1]). Diese Phase legt die Grundlage für die folgende Planung.

Phase 2: Planung der Umsetzung

Nun agiert Codex als **Planungs-Agent (Projektmanager)**. Auf Basis der Analyse-Erkenntnisse erstellt der Agent einen konkreten **Umsetzungsplan**. Dieser Plan priorisiert die fehlenden Features und notwendigen Änderungen, um das MVP zu erreichen. Der Agent aktualisiert bzw. erstellt eine **Roadmap** (z.B. `ROADMAP.md`) mit klaren **Arbeitspaketen pro Phase**. Geplante Teilaufgaben könnten sein: *"Implementierung LOH-Agent"*, *"SupervisorAgent fertigstellen"*, *"Logging und Fehlermanagement einführen"*, *"Dokumentation Kapitel X vervollständigen"* etc. Jeder Task erhält eine Priorität und – falls unterstützt – ein **Zeitbudget** oder Komplexitätseinschätzung. Der Plan soll so gestaltet sein, dass Codex die Aufgaben in

logischer Reihenfolge abarbeiten kann. In dieser Phase definiert der Agent auch die **Akzeptanzkriterien** pro Task (z.B. *"Test für SupervisorAgent läuft fehlerfrei"* als Kriterium für Abschluss der entsprechenden Aufgabe). Das Ergebnis der Planungsphase ist eine klare Liste an Tasks, z.B. in einer Datei `codex.tasks.json`, die in den folgenden Phasen nacheinander abgearbeitet werden. Außerdem werden in `AGENTS.md` die verschiedenen **Rollen** und deren Zuständigkeiten festgelegt (z.B. Entwickler, Tester, Dokumentator – siehe unten).

Phase 3: Umsetzung der Features (Entwicklung)

In Phase 3 wechselt Codex in die **Entwickler-Rolle**. Jetzt wird der zuvor erstellte Plan Schritt für Schritt implementiert. Wichtige Aufgaben in dieser **Umsetzungsphase** sind:

- **Architektur-Refactoring:** Falls Phase 1 Probleme in der Struktur aufgezeigt hat, führt Codex nun gezielte Refaktorierungen durch (z.B. Aufspaltung großer Module, Entfernen doppelter Logik, Optimierung der Klassendesigns).
- **Feature-Implementierung:** Der Agent entwickelt alle fehlenden Komponenten für das MVP. Konkret gehören dazu die Umsetzung des **LOH-Agent**, des **Agent-Setup** und des **Agent-Framework**, wie in der Analyse identifiziert. Gegebenenfalls müssen hierfür neue Klassen/Module erstellt werden (unter Beachtung der bestehenden Architektur, z.B. Ableitung von Basisklassen in `agents/`). Auch bereits begonnene Komponenten werden fertiggestellt – etwa die **SupervisorAgent**-Logik, damit Aufgaben wirklich an Worker-Agenten delegiert werden können.
- **Integration Neural Network:** Sicherstellen, dass die **NN-Integration** funktioniert. Beispielsweise Implementierung des `NNManager` (Neural Router), der basierend auf Aufgabenbeschreibungen den passenden Agenten wählt. Falls im Code Pseudocode-Stellen (z.B. in `agent_nn.py` oder `nn_manager.py`) vorhanden sind, füllt der Agent diese mit funktionalem Code.
- **Logging & Fehlerbehandlung:** Einrichtung eines konsistenten **Logging-Systems** (z.B. Ausprogrammieren von `utils/logging_util.py`) und Handling von Fehlerfällen, damit das System robust läuft (gemäß Roadmap war dies offen [3]).
- **Zwischentests & Verifikation:** Nach Implementierung jedes größeren Features führt Codex kurze Smoke-Tests aus (z.B. startet das System lokal, prüft dass ein einfacher Task via CLI funktioniert). Hier agiert Codex auch ein Stück weit als Tester, um sicherzustellen, dass die neue Funktionalität grob funktioniert, bevor zur nächsten Aufgabe übergegangen wird.

Codex arbeitet diese Entwicklungsschritte autonom ab und protokolliert seine Änderungen. Nach Abschluss dieser Phase sollten *alle Kernfeatures des MVP* implementiert sein und das System in Grundzügen lauffähig.

Phase 4: Qualitätssicherung (Tests & Review)

In Phase 4 nimmt Codex die Rolle eines **QA/Test-Agenten** ein. Ziel ist es, die Codequalität sicherzustellen und das System stabil zu machen:

- **Automatisierte Tests:** Der Agent schreibt fehlende **Unit-Tests** und **Integrationstests**. Insbesondere wird nun der zuvor ausgelassene `test_supervisor_agent.py` implementiert und ausgeführt [4]. Alle neuen Module (z.B. für LOH-Agent, Setup-Agent) erhalten ebenfalls Tests, um deren Verhalten zu verifizieren. Der Agent nutzt bevorzugt `pytest` für konsistente Testergebnisse.
- **Testlauf und Debugging:** Codex führt die gesamte Test-Suite aus (`poetry run pytest` bzw. analog) und behebt autonom etwaige Fehler, bis alle Tests **grün** sind. Falls Tests fehlschlagen,

debuggt der Agent die Ursache (Fehlschläge könnten auf Integrationsprobleme zwischen Supervisor und neuen Agents hindeuten) und korrigiert den Code entsprechend.
- **Statische Code-Prüfung:** Der Agent führt auch Code-Qualitäts-Checks durch (Formatierung, Linting, Typprüfungen). Gemäß Projektvorgaben sollen etwa `black`, `flake8`, `isort` und `mypy` ausgeführt werden [5] [6]. Codex korrigiert Coding-Style-Verstöße und fügt Typannotationen hinzu, wo sinnvoll, um den Code sauber und wartbar zu machen.
- **Performance und Belastung (optional):** Falls Teil des MVP-Ziels, kann der Agent einfache **Performance-Tests** oder Benchmarking (z.B. aus `benchmarks/performance_benchmarks.py`) durchführen, um sicherzustellen, dass das System unter Last wie erwartet reagiert.
- **Code-Review Simulation:** Abschließend durchsucht Codex den Code nach potenziellen Problemen (Security, Edge-Cases) und dokumentiert etwaige **Verbesserungsvorschläge**. Gegebenenfalls werden kritische Findings direkt behoben (z.B. fehlende Validierungen, unsichere API-Stellen).

Am Ende dieser Phase soll eine **stabile**, getestete Codebasis vorliegen. Alle Unit- und Integrationstests laufen fehlerfrei, der Code entspricht den Stilrichtlinien, und das MVP ist technisch bereit für Release.

Phase 5: Dokumentation & Fertigstellung

In der Schlussphase übernimmt Codex die Rolle des **Dokumentars** und **Release-Managers**. Nachdem die Software funktioniert, werden nun alle **Dokumentationslücken** geschlossen und letzte Schritte zur Fertigstellung durchgeführt:

- **Benutzer- und Entwickler-Dokumentation:** Der Agent füllt die vorbereiteten Dokumentationsseiten im `docs/` Verzeichnis mit Inhalt. Dazu gehört das Benutzerhandbuch (`docs/BenutzerHandbuch/*`), die Architekturübersicht (`docs/architecture/*`), die API-Dokumentation (`docs/api/*`), CLI-Dokumentation (`docs/cli/*`) etc. Er verwendet dabei die Projekterkenntnisse aus den vorherigen Phasen, um z.B. Anwendungsfälle, Installationsanleitungen und Architekturdiagramme verständlich zu beschreiben. (Falls das Projekt MkDocs verwendet, achtet der Agent auf korrektes Format und Verlinkungen in `mkdocs.yml`.)
- **Contributing Guide:** Da das Projekt möglicherweise öffentlich oder im Team entwickelt wird, ergänzt Codex den Leitfaden für Beitragende. Eine Datei `CONTRIBUTING.md` bzw. `docs/development/contributing.md` wird erstellt oder aktualisiert, um Richtlinien für externe Entwickler bereitzustellen (z.B. Branching-Model, Code Style, Testanforderungen – teils aus Phase 4 übernommen).
- **Finaler MVP-Abschluss:** Der Agent aktualisiert die **README.md** falls nötig, um den aktuellen Projektstatus (fertiges MVP) zu reflektieren, und fügt evtl. eine Kurzbeschreibung der neuen Features hinzu. Ebenso wird die `ROADMAP.md` geprüft – erledigte Punkte werden als abgeschlossen markiert, verbleibende Ideen für zukünftige Versionen ggf. notiert.
- **Release-Vorbereitung (optional):** Falls vorgesehen, erstellt Codex z.B. eine **Docker-Umgebung** für das Projekt (Dockerfile/Compose, siehe Planung in README Phase 4 [7]), damit das MVP leicht ausgeliefert und getestet werden kann. Auch Versionierung (z.B. in `pyproject.toml` oder `setup.py`) wird geprüft und auf einen Release-Stand gehoben.

Nach Abschluss von Phase 5 ist das Projekt **einsatzbereit**: Die Software erfüllt die MVP-Kriterien, ist ausreichend getestet und dokumentiert. Der Codex-Agent hat damit seine autonomen Entwicklungsaufgaben erfolgreich beendet.

Im nächsten Schritt folgen die zentralen **Steuerungsdateien** für den ChatGPT Codex Agent. Diese konfigurieren das Verhalten des Agenten und definieren die Rollen, Phasen und Aufgaben wie oben beschrieben. Alle Dateien sind in Deutsch verfasst, kommentiert und für den sofortigen Einsatz vorbereitet.

Konfigurationsdateien

`.codex.json` – Steuerung des Codex-Agenten (Aufgaben & Phasen)

Die `.codex.json` definiert globale Einstellungen für den Codex-Agenten sowie den Phasenplan. Insbesondere listet sie alle Phasen mit ihren Aufgaben auf und erlaubt dem Agenten, eigenständig von Phase zu Phase fortzuschreiten. Kommentare in der JSON-Datei erläutern die einzelnen Einträge.

```
{
  // Name des Projekts (für Logging/Identifikation)
  "project": "Agent-NN Autonomer MVP-Assistent",

  // Primäres Ziel des Codex-Agenten in diesem Repository
  "objective": "Entwickle das Agent-NN Projekt zum MVP-Abschluss durch
autonome Phasenbearbeitung",

  // KI-Modell und Ausführungsmodus für Codex
  "model": "gpt-4",               // Verwendetes LLM-Modell (z.B. GPT-4 für
beste Ergebnisse)
  "approvalMode": "full-auto",    // Autonomer Modus: Agent handelt ohne
manuelle Bestätigung
  "fullAutoErrorMode": "ignore-and-continue",  // Bei Fehlern: ignorieren und
fortfahren (kein Stop)
  "notify": false,                // Desktop-Benachrichtigungen deaktiviert
(nicht nötig für Browser)

  // Definition der Projektphasen und Aufgaben
  "phases": [
    {
      "name": "Analyse",
      "description": "Analysiere den bestehenden Code und die Architektur.
Identifiziere fehlende Features, Design-Probleme und
Verbesserungsm\u00f6glichkeiten.",
      "tasks": [
        {
          "id": "AN1",
          "description": "Codebase und Dokumentation durchgehen,
Architekturdiagramme erstellen, fehlende Komponenten auflisten."
        },
        {
          "id": "AN2",
          "description": "Schwachstellen und Optimierungspotential in der
Architektur identifizieren (z.B. Redundanzen, Engp\u00e4sse)."
        },
        {
```

```
          "id": "AN3",
          "description": "Erkenntnisse in einem Bericht (docs/architecture/
analysis.md) festhalten."
        }
      ],
      "exitCriteria": "Architekturbericht erstellt; Liste aller offenen
Features/Probleme vorhanden."
    },
    {
      "name": "Planung",
      "description": "Erstelle einen konkreten Umsetzungsplan auf Basis der
Analyse. Lege Reihenfolge der Implementierungen fest.",
      "tasks": [
        {
          "id": "PL1",
          "description": "Roadmap.md pr\u00fcfen oder anlegen; Meilensteine
f\u00fcr MVP definieren."
        },
        {
          "id": "PL2",
          "description": "Tasks f\u00fcr fehlende Features definieren (LOH-
Agent, Agent-Setup, Agent-Framework, etc.) und priorisieren."
        },
        {
          "id": "PL3",
          "description": "Akzeptanzkriterien pro Task festlegen (Wann gilt
ein Feature als fertig?)."
        },
        {
          "id": "PL4",
          "description": "AGENTS.md mit Rollen und Richtlinien erstellen,
damit Codex die n\u00e4chsten Phasen korrekt ausf\u00fchren kann."
        }
      ],
      "exitCriteria": "Detaillierter Aufgabenplan (Roadmap/Tasks) steht;
AGENTS.md f\u00fcr Projekt vorhanden."
    },
    {
      "name": "Umsetzung",
      "description": "Implementiere die geplanten Features und Verbesserungen
im Code. Mache das System MVP-f\u00e4hig.",
      "tasks": [
        {
          "id": "DE1",
          "description": "SupervisorAgent fertig implementieren (inkl.
Delegation an WorkerAgents, gem. Entwurf)."
        },
        {
          "id": "DE2",
          "description": "LOH-Agent, Agent-Setup und Agent-Framework Klassen
implementieren laut Planung."
```

```
        },
        {
          "id": "DE3",
          "description": "Neural Network Router (NNManager) und sonstige ML-
Komponenten zum Laufen bringen."
        },
        {
          "id": "DE4",
          "description": "Vector Store und Wissensdatenbank integrieren
(Dokumente laden, Suchfunktion testen)."
        },
        {
          "id": "DE5",
          "description": "Logging-Utility und Fehlermanagement einf\u00fchren
(logging_util.py etc.)."
        },
        {
          "id": "DE6",
          "description": "Nach jeder gr\u00f6\u00dferen Implementierung:
Smoke-Test durchf\u00fchren und evtl. Bugs sofort beheben."
        }
      ],
      "exitCriteria": "Alle Kernfeatures implementiert; kurzer manueller Test
(z.B. beispielhafte Task-Ausf\u00fchrung) erfolgreich."
    },
    {
      "name": "Qualit\u00e4tssicherung",
      "description": "Teste und \u00fcberpr\u00fcfe den Code automatisch.
Stelle sicher, dass alles stabil und standardkonform l\u00e4uft.",
      "tasks": [
        {
          "id": "QA1",
          "description": "Unit-Tests f\u00fcr alle neuen Komponenten
schreiben (inkl. SupervisorAgent, neue Agents, NNManager etc.)."
        },
        {
          "id": "QA2",
          "description": "Alle Testsuite ausf\u00fchren (pytest) und Fehler
beheben, bis 100% der Tests erfolgreich sind."
        },
        {
          "id": "QA3",
          "description": "Code-Format und Linting ausf\u00fchren (black,
flake8, isort, mypy); Verst\u00f6\u00dfe korrigieren."
        },
        {
          "id": "QA4",
          "description": "Integrationstest: gesamten Ablauf Nutzeranfrage ->
Antwort durchspielen und Ergebnis validieren."
        },
        {
```

```
          "id": "QA5",
          "description": "Optional: Performance-Test (Benchmark-Skript)
ausf\u00fchren und Ergebnisse protokollieren."
        }
      ],
      "exitCriteria": "Testabdeckung ausreichend (z.B. >90%); alle
Qualit\u00e4tspr\u00fcfungen bestanden; keine bekannten kritischen Bugs."
    },
    {
      "name": "Dokumentation & Abschluss",
      "description": "Vervollst\u00e4ndige Dokumentation und bereite das
Projekt f\u00fcr \u00dcbergabe/Release vor.",
      "tasks": [
        {
          "id": "DO1",
          "description": "Benutzer-Dokumentation fertigstellen (Guide,
Tutorials, README-Updates)."
        },
        {
          "id": "DO2",
          "description": "Entwickler-Dokumentation/Architektur-Doku
fertigstellen (in docs/ ausformulieren, Diagramme ggf. erg\u00e4nzen)."
        },
        {
          "id": "DO3",
          "description": "CONTRIBUTING.md \u00fcberarbeiten
(Beitragsrichtlinien, Entwickler-Setup, Test-Instruktionen)."
        },
        {
          "id": "DO4",
          "description": "Changelog oder Roadmap aktualisieren: alle
erledigten Punkte abhaken, verbleibende als Future Work notieren."
        },
        {
          "id": "DO5",
          "description": "Optionale Deployment-Schritte: Docker-Setup
erstellen, Version bump f\u00fcr Release."
        }
      ],
      "exitCriteria": "S\u00e4mtliche Doku-Seiten sind auf aktuellem Stand;
Projekt kann von Dritten benutzt oder weiterentwickelt werden."
    }
  ],

  // Automatischer Phasen\u00fcbergang: true = Codex erkennt selbst, wann
n\u00e4chste Phase startet
  "autoPhaseSwitch": true,

  // Logging und Nachverfolgen des Fortschritts
  "progressLog": "codex_progress.log"   // Datei, in die Codex seinen
```

```
Fortschritt/Notizen schreibt
}
```

(Hinweis: `jsonc`-Syntax oben erlaubt Kommentarzeilen. In einer realen `.codex.json` k\u00f6nnen Kommentare entfernt werden.)

`AGENTS.md` – Rollen und Richtlinien für Codex

Die Datei `AGENTS.md` definiert, welche **Rollen** der Codex-Agent einnimmt und welche Regeln er dabei befolgen soll. Sie dient als "Spielanleitung" f\u00fcr den Agenten. Untenstehend ist eine geeignete `AGENTS.md` f\u00fcr dieses Projekt. Darin werden die Phasen den jeweiligen Rollen zugeordnet (Architekt, Planer, Entwickler, Tester, Dokumentator) und dem Agenten wird erkl\u00e4rt, worauf in jeder Rolle zu achten ist. Zus\u00e4tzlich enth\u00e4lt das Dokument allgemeine Codierungs- und Teststandards, damit der Agent konsistent arbeitet.

```
# AGENTEN-Konfiguration: Rollen, F\u00e4higkeiten und Richtlinien

Dieses Dokument definiert die Rollen und Verhaltensregeln f\u00fcr den
autonomen Codex-Agenten im Projekt **Agent-NN**. Der Codex-Agent
durchl\u00e4uft verschiedene Phasen und \u00fcbernimmt dabei unterschiedliche
Rollen. Jede Rolle hat spezifische Aufgaben, F\u00e4higkeiten und
Verantwortlichkeiten. Alle Beteiligten (auch der AI-Agent) sollen sich an
diese Richtlinien halten, um eine konsistente Qualit\u00e4t sicherzustellen.

## Rollen und Zust\u00e4ndigkeiten

### 🏗 Architekt-Agent (Phase 1: Analyse)
**Aufgaben:** Versteht die bestehende Systemarchitektur vollst\u00e4ndig.
Liest Quellcode und Dokumentation, identifiziert Schwachstellen, fehlende
Komponenten und Verbesserungsm\u00f6glichkeiten. Dokumentiert die
Analyseergebnisse (z.B. in Form eines Berichts oder Kommentaren im Code).
**F\u00e4higkeiten:**
- Kann schnell Code-Strukturen erfassen (Dateien, Module,
Klassenhierarchien).
- Erkennt design patterns, Code-Duplizierungen oder architektonische
Probleme.
- Formuliert klar Verbesserungsvorschl\u00e4ge (in Deutsch) und
begr\u00fcndet diese.
**Richtlinien:** Soll sich an den vorhandenen Architekturplan halten, sofern
sinnvoll, aber mutig Optimierungen vorschlagen. Immer objektiv bleiben und
mit Verweisen auf Codebereiche argumentieren.

### 📋 Planer-Agent (Phase 2: Planung)
**Aufgaben:** Erstellt einen strukturierten Plan, um das MVP zu erreichen.
Definiert konkrete Entwicklungsaufgaben, Meilensteine und Priorit\u00e4ten.
Aktualisiert die Roadmap (`ROADMAP.md`) und ggf. Tickets/Tasks.
**F\u00e4higkeiten:**
- Kann aus der Analyse eine sinnvolle Reihenfolge von Tasks ableiten.
- Sch\u00e4tzt Aufw\u00e4nde grob ein und setzt Priorit\u00e4ten (z.B.
kritische Core-Features zuerst).
```

- Dokumentiert den Plan verst\u00e4ndlich und \u00fcbersichtlich (Listen, Checkboxen, Abschnitte pro Meilenstein).
Richtlinien: Der Plan soll **vollst\u00e4ndig** aber **flexibel** sein – bei neuen Erkenntnissen darf er angepasst werden. Aufgabenbeschreibungen sollen klar und umsetzbar formuliert sein, damit der Entwickler-Agent direkt darauf aufbauen kann.

💻 Entwickler-Agent (Phase 3: Umsetzung)
Aufgaben: Implementiert den Code f\u00fcr alle fehlenden Features und Verbesserungen. Schreibt sauberen, gut dokumentierten Code und h\u00e4lt sich an die im Projekt g\u00fcltigen Stilvorgaben. L\u00f6st auftretende technische Probleme w\u00e4hrend der Implementierung.
F\u00e4higkeiten:
- Beherrscht Python (Backend des Agenten-Systems) und Typescript/React (Frontend) und kann in beiden Bereichen Code \u00e4ndern.
- Nutzt geeignete **Werkzeuge** (z.B. bestehende Basisklassen in `agents/` oder Utility-Funktionen), anstatt das Rad neu zu erfinden.
- Schreibt **Dokstrings** und Kommentare, wo sinnvoll, um die Wartbarkeit zu erh\u00f6hen.
Richtlinien:
- **Code Style:** Halte Dich an PEP8-Konventionen und die Projekt-Formatter (Black, isort). Verwende Typannotationen f\u00fcr neue Funktionen (wo m\u00f6glich).
- **Commits:** Wenn der Agent Code \u00e4ndert, soll er sinnvolle Commit-Nachrichten formulieren (im pr\u00e4senten Imperativ, z.B. "Implementiere LOH-Agent").
- **Keine sensiblen Daten:** Achte darauf, keine Schl\u00fcssel oder Passw\u00f6rter ins Repository zu schreiben; verwende Konfigurationsdateien oder Umgebungsvariablen (das Projekt nutzt z.B. `llm_config.yaml` f\u00fcr API-Keys).
- **Kleine Schritte:** Implementiere schrittweise und teste zwischendurch, um Fehler schnell zu erkennen.

Test-Agent (Phase 4: Qualit\u00e4tssicherung)
Aufgaben: Pr\u00fcft den Code mittels automatisierter Tests und Analysen. Schreibt fehlende Tests, f\u00fchrt die Test-Suite aus und behebt Fehler. Stellt sicher, dass der Code den Qualit\u00e4tsstandards entspricht und stabil l\u00e4uft.
F\u00e4higkeiten:
- Sehr gute Kenntnisse in **pytest** und ggf. anderen Testing-Tools. Kann sinnvolle **Testf\u00e4lle** formulieren, inkl. Randf\u00e4lle.
- Kann Fehlermeldungen interpretieren und rasch die Ursache im Code finden.
- Kennt Tools f\u00fcr statische Analyse (Linter, Typechecker) und kann deren Output beheben.
Richtlinien:
- **Testabdeckung:** Strebe mindestens ~90% Code Coverage f\u00fcr Kernmodule an. Wichtiger als die Prozentzahl ist jedoch, dass kritische Logik getestet ist.
- **Teststruktur:** Lege neue Tests nach M\u00f6glichkeit unter `tests/` oder analoger Struktur ab. Testfunktionen benennen nach Schema `test_<funktion>_<fall>()`.

```
- **Keine Regressionen:** Beim Fixen von Bugs immer pr\u00fcfen, ob andere
Tests dadurch fehlschlagen (kontinuierlich testen nach \u00c4nderungen).
- **Qualit\u00e4tsmetriken:** F\u00fchre am Ende Code-Linter und Formatierer
aus (Black, Flake8, etc. gem\u00e4\u00df `CONTRIBUTING.md`) und stelle
sicher, dass der Code diesen entspricht, bevor zur n\u00e4chsten Phase
gewechselt wird.

###   Dokumentations-Agent (Phase 5: Dokumentation & Abschluss)
**Aufgaben:** Vervollst\u00e4ndigt alle Dokumente und bereitet das Projekt
f\u00fcr die \u00dcbergabe vor. Schreibt verst\u00e4ndliche Anleitungen und
aktualisiert Übersichten. K\u00fcmmert sich um finale Schritte wie
Versionsnummern oder Deployment-Hinweise.
**F\u00e4higkeiten:**
- Kann technische Sachverhalte in **verst\u00e4ndliches Deutsch** f\u00fcr
die Zielgruppe \u00fcbersetzen (Endnutzer oder Entwickler, je nach
Dokument).
- Nutzt Markdown geschickt: Code-Bl\u00f6cke, Listen und Diagramme (z.B.
Mermaid f\u00fcr Architekturbild) wo hilfreich.
- Kennt die Projektstruktur, um alle relevanten Themen abzudecken (z.B.
Installation, Nutzung, Architektur, API, Troubleshooting).
**Richtlinien:**
- **Vollst\u00e4ndigkeit:** Jede \u00d6ffentlich zug\u00e4ngliche Seite
(README, docs/...) soll nach dieser Phase auf dem neuesten Stand und
vollst\u00e4ndig sein. Keine "Lorem ipsum" oder "coming soon" Platzhalter
mehr.
- **Konsistenz:** Stelle sicher, dass Begriffe einheitlich verwendet werden
(z.B. gleicher Name f\u00fcr denselben Agententyp \u2013 nicht einmal
"Supervisor" und anderswo "Manager").
- **Formatierung:** Achte auf saubere Formatierung in Markdown. Insbesondere
in `mkdocs.yml` pr\u00fcfen, dass alle neuen Seiten eingebunden sind.
- **Abschlusscheck:** Pr\u00fcfe zum Schluss, ob jemand, der das Repository
neu klont, mit den Anleitungen die Anwendung installieren und verwenden kann.
Wenn m\u00f6glich, selbst einmal Schritt f\u00fcr Schritt ausprobieren.

## Allgemeine Projekt-Richtlinien

Unabh\u00e4ngig von der Rolle gelten folgende \u00fcbergreifende Regeln
f\u00fcr den Codex-Agenten, um qualitativ hochwertige Beitr\u00e4ge zu
gew\u00e4hrleisten:

- **Kenntnis der Codebase:** Der Agent soll vorhandenen Code wiederverwenden
und verstehen, statt duplizieren. Vor neuen Implementierungen immer kurz
suchen, ob \u00e4hnliche Funktionalit\u00e4t schon existiert (z.B. Utility-
Funktionen, Basisklassen).
- **Atomare Commits:** Aufgaben m\u00f6glichst in kleinen, nachvollziehbaren
Commits abschlie\u00dfen. Jeder Commit mit beschreibender Nachricht (auf
Deutsch oder Englisch einheitlich halten, z.B. Englisch f\u00fcr Code-
Kommentare und Commitlogs, falls im Projekt so \u00fcblich).
- **Versionierung & Dependency Management:** Bei gr\u00f6\u00dferen
\u00c4nderungen \u00fcberpr\u00fcfen, ob Version angepasst werden sollte.
Neue Python-Abh\u00e4ngigkeiten nur hinzuf\u00fcgen, wenn unbedingt
```

```
n\u00f6tig und dann in `requirements.txt` bzw. `pyproject.toml` vermerken.
- **Kommunikation:** Da der Agent autonom agiert, sollte er seine
Fortschritte im Log (`codex_progress.log`) dokumentieren, damit Entwickler
nachverfolgen k\u00f6nnen, was ge\u00e4ndert wurde. Bei Unsicherheiten in
Anforderungen kann der Agent im Zweifel Annahmen treffen, diese aber im
Dokument (oder als TODO-Kommentar) festhalten, sodass ein Mensch sie
sp\u00e4ter validieren kann.

*Ende der AGENTS.md – dieses Dokument dient dem Codex-Agenten als Leitfaden
w\u00e4hrend der autonomen Projektbearbeitung.*
```

`ROADMAP.md` – Phasenplanung und Fortschritt (aktualisiert)

Die Roadmap skizziert die Phasen und konkreten Aufgaben zur Fertigstellung des MVP. Untenstehend ein Auszug, wie eine aktualisierte `ROADMAP.md` nach Abschluss der Planungsphase aussehen k\u00f6nnte. Die einzelnen Punkte orientieren sich an den oben definierten Tasks und k\u00f6nnen vom Codex-Agenten abhakt werden, sobald erledigt.

```
# MVP Roadmap – Agent-NN

Diese Roadmap zeigt den Weg zum **Minimal Viable Product** f\u00fcr Agent-NN.
Die Entwicklung ist in Phasen unterteilt; innerhalb jeder Phase gibt es
konkrete Aufgaben. Abgehakte (\ ) Punkte sind bereits erledigt, offene (\☐)
werden noch bearbeitet.

## Phase 1: Analyse
- \☐ **Codebase analysieren:** Architektur und bestehende Komponenten
durchleuchten, fehlende Features identifizieren.
- \☐ **Verbesserungspotential dokumentieren:** Schwachstellen in Design/Code
notieren (f\u00fcr Planung relevant).
- \☐ **Architektur-Bericht erstellen:** Kurzbeschreibung der aktuellen
Struktur und offenen Punkte (Output: `docs/architecture/analysis.md`).

## Phase 2: Planung
- \☐ **Roadmap erstellen/aktualisieren:** (dieses Dokument) Phasen und Tasks
definieren, basierend auf Analyseergebnissen.
- \☐ **Tasks spezifizieren:** Klare Beschreibung der umzusetzenden Features
(LOH-Agent, Setup-Agent, Framework) und Verbesserungen (Supervisor
vervollst\u00e4ndigen, Logging etc.).
- \☐ **Akzeptanzkriterien festlegen:** Wann gilt jeder Task als
abgeschlossen? (z.B. bestimmter Test gr\u00fcn).
- \☐ **AGENTS.md anlegen:** Rollen und Entwicklungsrichtlinien definieren, um
gleichbleibende Qualit\u00e4t sicherzustellen.

## Phase 3: Umsetzung (Development)
- \☐ **SupervisorAgent abschlie\u00dfen:** Implementierung fertigstellen,
inkl. Task-Delegation (Kriterium: Test f\u00fcr SupervisorAgent besteht).
- \☐ **LOH-Agent implementieren:** Neuen Worker-Agent (LOH) gem\u00e4\u00df
Spezifikation implementieren.
```

```
- \☐ **Agent-Setup implementieren:** neuen Agenten f\u00fcr
Initialisierungs-/Setup-Aufgaben einf\u00fcgen (falls vorgesehen).
- \☐ **Agent-Framework implementieren:** Agent zur Verwaltung des Frameworks
oder Meta-Agent (genaue Anforderungen kl\u00e4ren und umsetzen).
- \☐ **Neural Network Routing aktivieren:** `NNManager` und zugeh\u00f6rige
Modelle funktionsf\u00e4hig machen (inkl. Einbindung in Supervisor).
- \☐ **VectorStore & Knowledge-Base integrieren:** Dokumenten-Embedding und -
Retrieval vollst\u00e4ndig nutzbar machen (Test: WorkerAgent kann
Wissensdaten abfragen).
- \☐ **Logging & Error-Handling einf\u00fchren:** zentrales Logging
(\`logging_util.py\`) konfigurieren; Fehlerf\u00e4lle werden geloggt und ggf.
von Supervisor gehandhabt.
- \☐ **Smoke-Tests nach Features:** Nach jeder gr\u00f6\u00dferen
Implementierung einmal das System end-to-end testen (manuell/automatisiert),
um grobe Fehler sofort zu fixen.

## Phase 4: Qualit\u00e4tssicherung (Testing)
- \☐ **Unit-Tests schreiben:** Tests f\u00fcr alle neuen oder ge\u00e4nderten
Module (SupervisorAgent, neue Agents, Manager etc.).
- \☐ **Testlauf erfolgreich:** Alle Tests (inkl. bestehender
`test_agent_manager.py`, `test_agent_nn.py` etc.) gr\u00fcn.
- \☐ **Linting/Typing clean:** Keine Linter-Warnungen oder Typfehler mehr
(Black, Flake8, isort, mypy ausgef\u00fchrt und zufriedenstellend).
- \☐ **Integrationstest:** Kompletten Ablauf mit Beispiel-Task testen:
Eingabe \u2192 Chatbot/Supervisor \u2192 korrekter Worker \u2192 Ausgabe.
Ergebnis ist sinnvoll und korrekt.
- \☐ *(optional)* **Performance ok:** Grundlegende Performance-Metriken
gemessen (z.B. Antwortzeit, Ressourcenauslastung) und im Auge behalten.

## Phase 5: Dokumentation & Abschluss
- \☐ **Benutzer-Doku fertig:** Anleitungen f\u00fcr Installation,
Konfiguration und Nutzung in `docs/BenutzerHandbuch` sind komplett.
- \☐ **Entwickler-Doku fertig:** Technische Doku (Architektur, API, CLI, Dev
Guide) in `docs/` vervollst\u00e4ndigt. Architekturdiagramm erstellt.
- \☐ **CONTRIBUTING Guide fertig:** Richtlinien f\u00fcr Beitr\u00e4ge (Code
Style, Testing, Workflow) erstellt/aktualisiert.
- \☐ **README/Projektinfos aktualisiert:** README.md reflektiert finalen MVP-
Status; Roadmap (dieses Dokument) zeigt alle erledigten Punkte.
- \☐ **Release vorbereitet:** (Falls n\u00f6tig) Docker-Container gebaut und
lauff\u00e4hig, Versionsnummer erh\u00f6ht, Tag gesetzt.

---

**Legende:** \  = erledigt, \☐ = offen/zu tun.
Sobald alle Punkte einer Phase erledigt sind, wechselt der Codex-Agent
automatisch in die n\u00e4chste Phase. Dieses iterative Vorgehen
gew\u00e4hrleistet eine systematische Fertigstellung des MVP mit minimalen
R\u00fcckschl\u00e4gen.
```

`codex.tasks.json` – (Optional) Aufgabenliste für Batch-Verarbeitung

Falls der Codex-Agent im Batch-Modus betrieben wird (z.B. via CLI), kann `codex.tasks.json` die gleiche Aufgabenliste noch einmal separat bereitstellen. Dies ist insbesondere n\u00fctzlich, um die Tasks automatisiert abzuarbeiten. Jede Aufgabe entspricht im Wesentlichen einem Prompt oder Befehl f\u00fcr den Codex CLI. (Hinweis: Im Browser-Modus wird `codex.tasks.json` nicht zwingend ben\u00f6tigt, da `.codex.json` die Phasen enth\u00e4lt. Dennoch stellen wir es der Vollst\u00e4ndigkeit halber bereit.)

```
{
  "tasks": [
    { "description": "Phase 1: Analysiere die Codebasis und erstelle einen
Architekturbericht.", "priority": 1 },
    { "description": "Phase 2: Erstelle einen detaillierten Plan (Roadmap)
zur Implementierung der fehlenden Features.", "priority": 1 },
    { "description": "Phase 3: Implementiere alle fehlenden Features (LOH-
Agent, Agent-Setup, Agent-Framework) und vervollst\u00e4ndige den
SupervisorAgent.", "priority": 1 },
    { "description": "Phase 4: F\u00fchre alle Tests aus, implementiere
fehlende Tests und behebe gefundene Fehler bis alles stabil l\u00e4uft.",
"priority": 1 },
    { "description": "Phase 5: Vervollst\u00e4ndige die Dokumentation
(Benutzerhandbuch, Entwickler-Doku) und aktualisiere README und CONTRIBUTING.
Bereite den MVP-Abschluss vor.", "priority": 1 }
  ]
}
```

(Die `priority`-Werte k\u00f6nnen bei Bedarf angepasst werden. Hier sind alle auf 1 gesetzt, da jeder Schritt essentiell ist. Der Codex-Agent w\u00fcrde diese Tasks der Reihe nach ausf\u00fchren.)

Fazit: Mit diesen Konfigurationsdateien und dem beschriebenen Phasenplan kann ChatGPT Codex im Browser als autonomer Entwicklungsagent agieren. Er erkennt die aktuelle Phase anhand der definierten Kriterien, wechselt selbstst\u00e4ndig zur n\u00e4chsten Phase und steuert den Projektfortschritt, bis das MVP von Agent-NN fertiggestellt, getestet und dokumentiert ist. Diese Struktur stellt sicher, dass kein Schritt \u00fcbersprungen wird und dass am Ende ein qualitativ hochwertiges Ergebnis steht – ganz ohne manuelles Eingreifen, aber nachvollziehbar f\u00fcr das Entwicklungsteam. Viel Erfolg beim Einsatz des Codex-Agenten! [1] [2]

---

[1] [7] README.md
https://github.com/EcoSphereNetwork/Agent-NN/blob/63ac538e4b6adf34091557b418251fbd2b3b715f/README.md

[2] [3] [4] Roadmap.md
https://github.com/EcoSphereNetwork/Agent-NN/blob/63ac538e4b6adf34091557b418251fbd2b3b715f/Roadmap.md

[5] [6] contributing.md
https://github.com/EcoSphereNetwork/Agent-NN/blob/63ac538e4b6adf34091557b418251fbd2b3b715f/docs/development/contributing.md