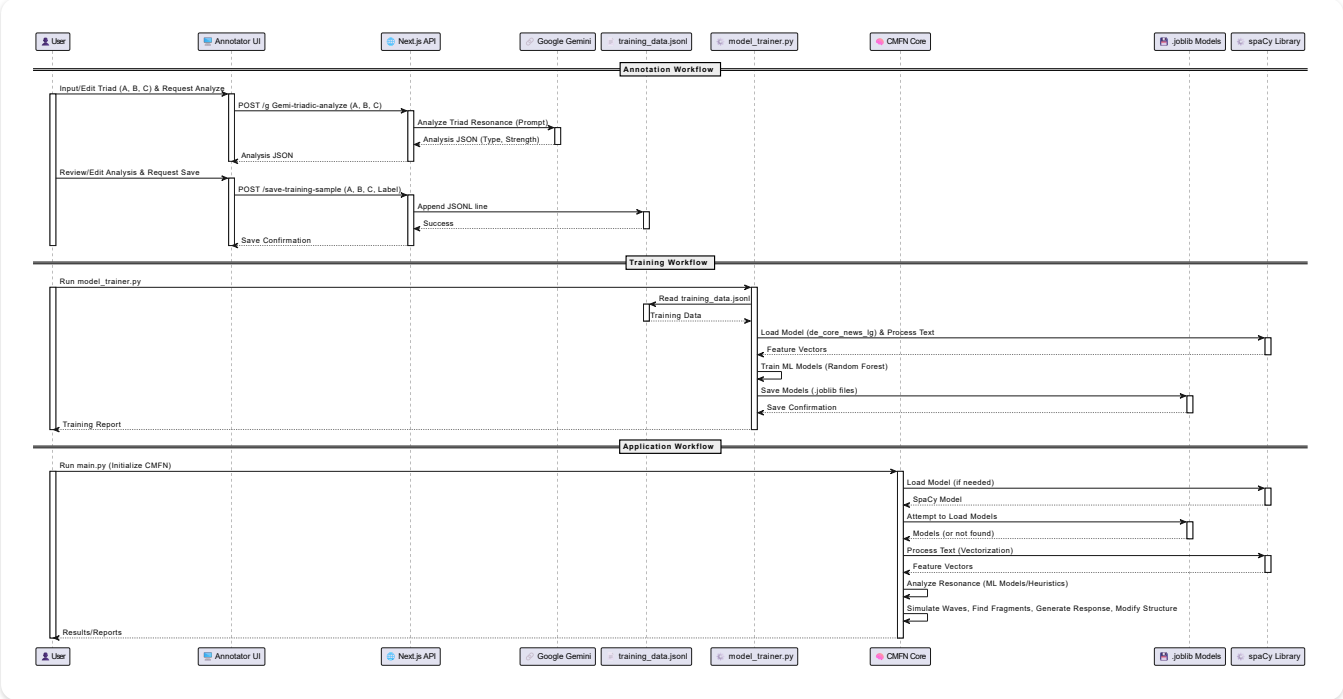
```html

# Dokumentation: Das Gewebe des Verstehens

## 1. Übersicht des Projekts

Das Projekt "Das Gewebe des Verstehens" ist ein Prototyp eines neuartigen KI-Systems zur Analyse und Simulation von Textverständnis, basierend auf dem Konzept eines **Contextual Meaning Field Network (CMFN)**. Im Gegensatz zu traditionellen Modellen, die auf sequenzieller Token-Vorhersage basieren, modelliert das CMFN Bedeutung als ein dynamisches Feld von Beziehungen (Resonanzen) zwischen semantischen Einheiten, repräsentiert als Knoten in einem Graphen.

Ein zentrales Merkmal ist die explizite Modellierung der **triadischen Resonanz**: der Einfluss eines dritten Textfragments (C) auf die bestehende Beziehung zwischen zwei anderen (A und B). Diese "Spürlogik" wird durch einen hybriden Ansatz umgesetzt, der eine robuste heuristische Basis mit einem datengestützten Machine-Learning-Modell (Random Forest) kombiniert.

Das Projekt gliedert sich in zwei Hauptteile:

- **`python_gewebe`**: Die Kernimplementierung des CMFN in Python, die das dynamische Graphenmodell, die ML-gestützte/heuristische Spürlogik, Resonanz-Wellen-Simulationen und generative Fähigkeiten beherbergt.
- **`nextjs_annotator`**: Ein Web-basiertes Frontend (Next.js), das speziell für die Generierung und Kuration von Trainingsdaten für die triadische Resonanzanalyse entwickelt wurde. Es nutzt die Google Gemini API, um erste Analysen vorzuschlagen, die dann von einem menschlichen Experten überprüft und korrigiert werden können ("Human-in-the-Loop").

Der Workflow ermöglicht es, durch Annotationen die "Spürlogik" des Python-Backends kontinuierlich zu verbessern und anzupassen, was zu einem System führt, das komplexe, kontextabhängige und assoziative Textzusammenhänge erfassen kann.

## 2. Architekturdiagramm

Das folgende Sequenzdiagramm veranschaulicht die Interaktion zwischen den Hauptkomponenten des Systems während der Annotation, des Trainings und der Anwendung.
```



3. Komponentenübersicht

Das Projekt ist in zwei Hauptverzeichnisse unterteilt, die die Kernkomponenten enthalten:

3.1. `python_gewebe/` (Kernlogik)

- **`main.py`**: Das Hauptskript zur Demonstration der CMFN-Funktionalität. Initialisiert das Gewebe, fügt Fragmente hinzu, simuliert Reaktionen auf Impulse, sucht nach Resonanzen und demonstriert generative Fähigkeiten. Lädt beim Start automatisch trainierte Modelle.
- **`text_gewebe.py`**: Enthält die Kernklasse (`NeuesTextVerstehen`), die das dynamische Graphenmodell (Fragmente als Knoten, Resonanzen als Kanten), die Logik für dyadische und triadische Resonanzanalyse (mit ML-Fallback), Resonanz-Wellen und die Verwaltung der Gewebestruktur (Hinzufügen, Löschen, Verschmelzen) implementiert.
- **`model_trainer.py`**: Ein eigenständiges Skript zum Trainieren der Machine-Learning-Modelle für die triadische Resonanzanalyse. Es liest Daten aus `training_data.jsonl`, nutzt spaCy zur Feature-Extraktion und trainiert Random Forest Klassifikator- und Regressionsmodelle, die als `.joblib`-Dateien gespeichert werden.
- **`training_data.jsonl`**: Die Datei, in der die vom Annotator generierten und kuratierten Trainingsdaten im JSON Lines Format gespeichert werden.
- **`.joblib` Dateien**: Gespeicherte, trainierte ML-Modelle (`triadic_art_classifier.joblib`, `triadic_staerke_regressor.joblib`) und der Label Encoder (`triadic_label_encoder.joblib`), die von `text_gewebe.py` geladen werden.

3.2. `nextjs_annotator/` (Annotationstool)

- **`app/page.tsx`**: Die Hauptseite des Next.js-Frontends, die das Benutzerinterface für die Eingabe von Fragmenten, das Starten der Gemini-Analyse, die Überprüfung/Bearbeitung

der Ergebnisse und das Speichern der Trainingsdaten bereitstellt.
- **`app/api/gemini-generate-triads/route.ts`**: Eine API-Route, die Anfragen an Google Gemini sendet, um Vorschläge für neue Text-Triaden (A, B, C) basierend auf einem Thema zu generieren.
- **`app/api/gemini-triadic-analyze/route.ts`**: Eine API-Route, die Anfragen an Google Gemini sendet, um die triadische Resonanz (Art und Stärke) für eine gegebene Triade (A, B, C) zu analysieren.
- **`app/api/save-training-sample/route.ts`**: Eine API-Route, die die annotierten Daten empfängt und sie als neue Zeile an die Datei `python_gewebe/training_data.jsonl` anhängt.
- **`components/ui/`**: Enthält UI-Komponenten, die mit Shadcn UI und Tailwind CSS erstellt wurden (z.B. Button, Card, Input, Textarea, Label).
- **`lib/utils.ts`**: Hilfsfunktionen, insbesondere für die Handhabung von Tailwind CSS Klassen (`cn`).

4. Wichtige Abhängigkeiten

Die Kernfunktionalität des Projekts basiert auf den folgenden Bibliotheken und Frameworks:

Python-Backend (`python_gewebe`)

- **spaCy**: Wird für die Verarbeitung natürlicher Sprache, insbesondere für die Vektorisierung von Textfragmenten (`de_core_news_lg` Modell), verwendet.
- **scikit-learn**: Bietet die Machine-Learning-Algorithmen (Random Forest Classifier und Regressor) für das Training der triadischen Resonanzmodelle.
- **joblib**: Wird zum Speichern und Laden der trainierten scikit-learn Modelle verwendet.
- **numpy**: Wird für numerische Operationen und die Handhabung von Vektoren und Arrays benötigt.

Next.js-Frontend (`nextjs_annotator`)

- **Next.js**: Das React-Framework für das Web-Frontend, das Server-Side Rendering und API-Routen ermöglicht.
- **React**: Die JavaScript-Bibliothek für den Aufbau der Benutzeroberfläche.
- **@google/generative-ai**: Die offizielle Bibliothek zur Interaktion mit der Google Gemini API.
- **Tailwind CSS**: Ein Utility-First CSS Framework für schnelles Styling.
- **Shadcn UI**: Eine Sammlung von wiederverwendbaren UI-Komponenten, die mit Radix UI und Tailwind CSS erstellt wurden.

5. Einrichtung und Nutzung

Befolgen Sie diese Schritte, um das Projekt einzurichten und den Workflow zu durchlaufen:

5.1. Teil 1: Python-Backend (`python_gewebe`)

1. Navigieren Sie in das Python-Verzeichnis:

   ```
   cd python_gewebe
   ```

2. Erstellen und aktivieren Sie eine virtuelle Umgebung (empfohlen):

   ```
   python -m venv venv
   ```

   Auf macOS/Linux:

   ```
   source venv/bin/activate
   ```

   Auf Windows:

   ```
   venv\Scripts\activate
   ```

3. Installieren Sie die Abhängigkeiten:

   ```
   pip install -r requirements.txt
   ```

4. Laden Sie das spaCy-Sprachmodell herunter:

   ```
   python -m spacy download de_core_news_lg
   ```

5.2. Teil 2: Next.js-Frontend (`nextjs_annotator`)

5. Navigieren Sie in das Next.js-Verzeichnis:

   ```
   cd nextjs_annotator
   ```

6. Erstellen Sie eine `.env.local`-Datei im `nextjs_annotator`-Verzeichnis und fügen Sie Ihren Google Gemini API-Schlüssel hinzu:

   ```
   GEMINI_API_KEY="IHR_GOOGLE_AI_STUDIO_API_KEY"
   ```

   Ersetzen Sie `"IHR_GOOGLE_AI_STUDIO_API_KEY"` durch Ihren tatsächlichen Schlüssel.

7. Installieren Sie die Node.js-Abhängigkeiten:

   ```
   npm install
   ```

5.3. Workflow: Vom Annotieren zum trainierten Modell

Der Prozess besteht aus drei Schritten:

Schritt 1: Daten annotieren

1. Starten Sie den Annotator im `nextjs_annotator`-Verzeichnis:

   ```
   npm run dev
   ```

2. Öffnen Sie das UI im Browser unter http://localhost:3000.
3. Nutzen Sie das UI, um Textfragmente (A, B, C) einzugeben oder generieren zu lassen, die Analyse mit Gemini zu starten, die Ergebnisse zu überprüfen/korrigieren und als Trainingsdaten zu speichern. Jede Speicherung fügt eine Zeile zu `python_gewebe/training_data.jsonl` hinzu. Sammeln Sie genügend Daten (mindestens 10-20, idealerweise 50+).

Schritt 2: Modelle trainieren

4. Navigieren Sie zurück in das Python-Verzeichnis (`python_gewebe`).
5. Stellen Sie sicher, dass Ihre virtuelle Umgebung aktiviert ist.
6. Führen Sie das Trainingsskript aus:

   ```
   python model_trainer.py
   ```

   Das Skript liest die Daten, trainiert die Modelle und speichert sie als `.joblib`-Dateien im selben Verzeichnis.

Schritt 3: Das trainierte Gewebe anwenden

7. Bleiben Sie im `python_gewebe`-Verzeichnis.
8. Führen Sie das Hauptskript aus:

   ```
   python main.py
   ```

   Das Skript initialisiert das CMFN. Wenn die `.joblib`-Modelle vorhanden sind, werden diese geladen und für die triadische Resonanzanalyse verwendet. Andernfalls wird der heuristische Fallback genutzt. Das Skript demonstriert dann verschiedene CMFN-Funktionen.

```
```