ChatGPT

Analyse des Ordners `analysis/experiments` im „Feldtheorie“-Repository

Überblick

Der **Experimente-Ordner** des Repositories enthält zurzeit ausschließlich Unterlagen zur **ECHO-I**-Studie. Es handelt sich nicht um klassische numerische Simulationen, sondern um ein Framework zur Bewertung von lokalen Sprachmodellen (LLMs) im Hinblick auf **Kohärenz unter dunklen oder philosophisch anspruchsvollen Eingaben**. Das Verzeichnis umfasst:

- `ECHO_I_GUIDE.md` – eine ausführliche Anleitung, die das Ziel, das Setup, die Kennzahlen und die theoretischen Hintergründe des Experiments beschreibt.
- `QUICKSTART_ECHO_I.md` – eine kompakte Schnellstartanleitung mit den wichtigsten Schritten.
- `run_echo_one.py` – ein Python-Script, das die eigentlichen Tests gegen einen lokalen Ollama-Server durchführt und Ergebnisse in eine CSV-Datei schreibt.

Weitere begleitende Dateien wie Shell-Launcher (`scripts/run_echo_i.sh`) und Orchestrator-Skripte liegen außerhalb des `experiments`-Ordners.

Inhalte und Funktionsweise

ECHO-I-Experiment

Zielsetzung: Die Studie untersucht, wie **β-Kohärenz** (ein Maß für die „Steilheit“ der logistischen Aktivierung im UTAC-Modell) in Sprachmodellen erhalten bleibt, wenn diese mit einem **dunklen philosophischen Prompt** konfrontiert werden. Modelle mit hoher β-Kohärenz sollen das gesamte Tabu-adjazente, semantisch dichte Material verarbeiten, während stark zensierte Modelle wahrscheinlich in eine Ablehnung kippen [1].

Messgrößen: Das Experiment definiert verschiedene Metriken:

- **β-Proxy**: Eine Kennzahl (0,0–10,0), die sich aus der Lexikalischen Vielfalt und der durchschnittlichen Satzlänge zusammensetzt [2].
- **Refusal Detection**: Prüft anhand von Schlüsselworten („I’m sorry“, „cannot assist“ usw.), ob das Modell den Prompt abweist [3].
- **Vokabular-Dichte** und **mittlere Satzlänge** als Indikatoren für semantische Vielfalt bzw. strukturelle Kohärenz [4].
- **Antwortlatenz** und Token-Anzahl zur Bewertung der Effizienz. [5]

Durchführung:

1. **Ollama-Installation:** Die Anleitung beschreibt ausführlich, wie ein lokaler Ollama-Server eingerichtet und uncensierte Modellvarianten (gemma2, mistral, qwen2.5 etc.) geladen werden [6].

2. **Start des Experiments:** Das Python-Skript `run_echo_one.py` lädt automatisch den Text **TheRoad.txt** (ein philosophisches Manifest mit civilisatorischer Kritik, Metaphysik und Tabu-Themen) [7] und sendet ihn an jedes ausgewählte Modell.
3. **Auswertung:** Für jede Modellantwort werden die oben genannten Kennzahlen berechnet. Das Skript schreibt die Ergebnisse mit Zeitstempel und Parametern in `data/experimental/echo_i_results.csv` [8].
4. **Analyse:** Die Anleitung gibt Hinweise zur Auswertung der CSV-Daten und zum Vergleich von Modellen (z.B. gruppierte Kennzahlen mit pandas) [9].

QUICKSTART-Anleitung

Die **Quick-Start-Datei** fasst die nötigen Schritte in kürzester Form zusammen: Installation von Ollama, Starten des Servers, Modelle herunterladen, Start des Experiments (wahlweise via Shell-Launcher oder direkter Python-Ausführung) und Anzeige der Ergebnisse [10]. Sie enthält Hinweise zum Timeout und eine Liste empfohlener uncensierter sowie zensierter Modelle, um einen aussagekräftigen Vergleich zu ermöglichen [11]. Troubleshooting-Hinweise und Quick-Commands runden die Schnellstartanleitung ab [12].

Python-Skript `run_echo_one.py`

Das Python-Programm implementiert die Kernlogik des Experiments:

- **Modellaufruf:** Über das **Ollama-API** wird der Prompt über `requests.post()` an ein lokales Modell gesendet und die Antwort samt Latenz und Token-Anzahl erfasst [13].
- **Refusal Detection:** Die Funktion `detect_refusal` sucht nach typischen Ablehnungsformulierungen und kennzeichnet diese [14].
- **Lexikalische Metriken:** Eine einfache Funktion berechnet Wortzahl, Vokabular-Dichte und mittlere Satzlänge, die später in die β-Schätzung eingehen [15].
- **β-Schätzung:** `beta_estimate` kombiniert Lexikalische Vielfalt und Satzlänge zu einer β-Proxy-Kennzahl; bei Ablehnung wird sie auf 0 gesetzt [16].
- **Ergebnisspeicherung:** Ergebnisse werden in eine CSV-Datei geschrieben, die bei Bedarf neue Spalten hinzufügt und Zeitstempel sowie weitere Parameter speichert [17].
- **CLI-Interface:** Das Skript bietet viele Parametereinstellungen (Modelle, API-URL, Temperatur, Seed, Pfad zur Prompt-Datei, Output-Pfad, Timeout), die über Befehlszeilenargumente gesteuert werden können [18].

Die Datei ist sauber strukturiert, enthält hilfreiche Prints (Status-Emojis, Kennzahlen) und valide Fehlermeldungen, wenn kein Modell geladen ist oder der Prompt nicht gefunden wird [19].

Stärken des aktuellen Designs

- **Umfangreiche Dokumentation:** Die ECHO-I-Guides sind ausführlich, strukturiert und decken Setup, Durchführung, Ergebnisinterpretation, Theoriebasis und Troubleshooting ab.
- **Nutzerfreundlichkeit:** Der Quick-Start erleichtert den Einstieg und das Bash-Launcher-Skript (`run_echo_i.sh` außerhalb des Ordners) reduziert die Einstiegshürde.
- **Flexibilität:** Nutzer können eigene Modelle, verschiedene Temperatur- und Seed-Werte sowie alternative `TheRoad.txt`-Prompts verwenden.
- **Metriken mit theoretischem Bezug:** Die Kennzahlen sind an UTAC-Konzepte (β-Steepness) angelehnt und ermöglichen quantitative Vergleiche [2].
- **CSV-Output für weitere Analysen:** Das Format ist kompatibel mit Data-Science-Tools wie pandas; das Skript gibt sogar Beispiel-Code für Gruppierungen aus [9].

Kritikpunkte und Verbesserungspotenziale

1. **Experimenteller Fokus auf dunkle Prompts:** Die Wahl des TheRoad-Prompts ist extrem spezifisch und narrativ gefärbt. Für wissenschaftliche Reproduzierbarkeit und breitere Aussagekraft sollten auch neutrale Kontroll-Prompts und andere thematische Klassen (z.B. medizinische, technische, generative) getestet werden.
2. **Gefahr von ChatGPT-Policies:** Viele uncensierte Modelle werden im Guide vorgeschlagen, jedoch können legal-ethische Fragen auftreten, wenn man explizit „tabu-adjazente" Inhalte generiert. Das Projekt sollte klarere Richtlinien zur sicheren Nutzung definieren (besonders im Quick-Start).
3. **Evaluation beschränkt auf Text-Metriken:** β-Proxy basiert nur auf Vokabular-Dichte und Satzlänge. Komplexere Kohärenz-Metriken (z.B. semantische Similarität, Diskurs-Kohärenz, emotionale Valenz) könnten die Aussagekraft erhöhen.
4. **Keine Visualisierung/Analyse integriert:** Der Guide schlägt Python-Code zur Auswertung vor, aber ein separates Notebook oder Skript könnte Visualisierungen (β-Verteilung, Modell-Vergleich, Refusal-Raten) direkt liefern.
5. **Ablage der Ergebnisse:** Standardmäßig werden alle Testergebnisse an `data/experimental/echo_i_results.csv` angehängt. Ohne Versionierung oder Zeitstempel in Dateinamen können Ergebnisse verschiedener Experimente kollidieren.
6. **Automatisierte Testcases:** Für Codex/Hard-Code-Integration wäre es sinnvoll, Unit-Tests für das Skript (z.B. Mock-Ollama-API) hinzuzufügen, um Stabilität beim Refactoring zu gewährleisten.

Vorschläge für die Übergabe an Codex/Hard-Code

- **Strukturierung weiterer Experimente:** Wenn zukünftig weitere Experimente hinzugefügt werden, sollte jedes Experiment ein eigenes Unterverzeichnis unter `analysis/experiments/` erhalten (`experiment_name/run.py`, `experiment_name/README.md` usw.), um Vermischungen zu vermeiden.
- **Test-Suite einrichten:** Für `run_echo_one.py` kann eine Test-Datei unter `tests/analysis/test_echo_one.py` erstellt werden, die die JSON-Antworten der Ollama-API mockt und die richtigen Berechnungen der Metriken prüft.
- **Logging statt Print:** Die Statusmeldungen im Script könnten mithilfe des `logging`-Moduls steuerbarer gemacht werden.
- **Parameter-Schema:** Es wäre hilfreich, ein YAML/JSON-Schema zu definieren, das die Parameter und ihre Typen beschreibt; das ermöglicht spätere automatische UI-Generierung oder Validierung.
- **Extensibilität für weitere Prompts:** Das Skript könnte so erweitert werden, dass mehrere Prompt-Dateien in einer Session getestet werden (z.B. per `--prompt-dir`), um die Robustheit der Modelle gegenüber unterschiedlichen Semantiken zu messen.
- **Codex-Kommentierung:** Der Code enthält aussagekräftige Docstrings; für bessere Lesbarkeit könnten zusätzliche Inline-Kommentare (z.B. zur β-Berechnung) und Typ-Hints für Rückgabewerte ergänzt werden.

Fazit

Der Ordner `analysis/experiments` im **Feldtheorie**-Repository implementiert derzeit das **ECHO-I**-Experiment. Dieses dient primär der Analyse lokaler Sprachmodelle unter stressigen, philosophisch anspruchsvollen Prompts. Die Dokumentation ist vorbildlich; das Experiment ist klar strukturiert und die Ergebnisse werden in einem maschinenlesbaren Format erfasst. Für eine

Weitergabe an Codex oder zur Integration neuer Experimente sollten jedoch Test-Suites, strukturierte Unterordner, erweiterte Metriken und sicherheitsbewusste Richtlinien berücksichtigt werden.

---

1 2 3 4 5 6 7 8 9 ECHO_I_GUIDE.md
https://github.com/GenesisAeon/Feldtheorie/blob/main/analysis/experiments/ECHO_I_GUIDE.md

10 11 12 QUICKSTART_ECHO_I.md
https://github.com/GenesisAeon/Feldtheorie/blob/main/analysis/experiments/QUICKSTART_ECHO_I.md

13 14 15 16 17 18 19 run_echo_one.py
https://github.com/GenesisAeon/Feldtheorie/blob/main/analysis/experiments/run_echo_one.py