Verstanden. Unten bekommst du drei Dinge, sauber getrennt:

1. eine präzise **Installations- und Setup-Anleitung (WSL-fokussiert)**,
2. eine **kompakte, belastbare Befehls-/Optionen-Referenz** inkl. Speicherpfade,
3. eine fertig nutzbare **AGENTS.md** mit den von dir gewünschten Rollen (Core/Advanced/Specialized) – erweitert um das Nötige für **Python** sowie **Flutter (Android/iOS)**.

Alle Aussagen sind auf offizielle Quellen gestützt; wo OpenAI die Optionen aktiv weiterentwickelt, weise ich darauf hin.

**1) Installation & Setup (WSL empfohlen)**

**Voraussetzungen (WSL/Ubuntu 22.04+):**

* Windows 11 **über WSL2**, alternativ native macOS/Linux. ([GitHub](https://github.com/openai/codex))
* Node.js ≥ 20, Git empfohlen. ([GitHub](https://github.com/openai/codex))

**Schritte:**

# In WSL/Ubuntu

# (Optional) Node via NVM:

# curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

# source ~/.nvm/nvm.sh && nvm install --lts

npm install -g @openai/codex # CLI global installieren

codex --version # Installation prüfen

Quelle & Quickstart: OpenAI Help Center und GitHub-README. ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started), [GitHub](https://github.com/openai/codex))

**Anmeldung / Abrechnung:**

* **Sign-in mit ChatGPT:** codex --login (neuer Flow; Plus/Pro unterstützt, Promo-Credits via codex --free). ([OpenAI Help Center](https://help.openai.com/en/articles/11381614-codex-cli-and-sign-in-with-chatgpt))
* **Alternativ API-Key:** export OPENAI\_API\_KEY="…", dann codex. ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started))

**Erster Start (im Repo-Ordner):**

cd /mnt/d/…/dein-projekt

codex # TUI, "Suggest"-Modus

# oder:

codex --auto-edit # Dateien automatisch schreiben, Kommandos mit Rückfrage

codex --full-auto # Lesen/Schreiben/Kommandos autonom in Sandbox

Approval-Modi & Bedeutungen siehe Help-Artikel (Suggest/Auto-Edit/Full-Auto). ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started))

**Sandbox & Autonomie feinsteuern (empfohlen):**

# Empfohlene, sichere Defaults für Git-Repos:

codex --sandbox workspace-write --ask-for-approval on-request

# vollständig ohne Rückfragen (vorsichtig!):

codex --ask-for-approval never --sandbox read-only

Details (Werte & Presets) stehen im README „Choosing Codex’s level of autonomy“. ([GitHub](https://github.com/openai/codex))

**Konfigurationsdatei anlegen (TOML):** ~/.codex/config.toml

# Standardmodell und Grundverhalten

model = "gpt-5" # Default kann je Version variieren

approval\_policy = "on-request" # Rückfragen, wenn sinnvoll

sandbox\_mode = "workspace-write" # im Workspace schreiben, sonst fragen

# Profile (umschaltbare Voreinstellungen)

[profiles.full\_auto]

approval\_policy = "on-request"

sandbox\_mode = "workspace-write"

[profiles.readonly\_quiet]

approval\_policy = "never"

sandbox\_mode = "read-only"

# Optional: Netzwerk für workspace-write erlauben

[sandbox\_workspace\_write]

network\_access = false

# OSS-Provider (z.B. Ollama)

[model\_providers.oss]

name = "Open Source"

base\_url = "http://localhost:11434/v1"

– Konfig-Pfad, Profile & Sandbox-Feinsteuerung: GitHub-README.  
– Default-Modell: laut aktuellen Releases „gpt-5“; ältere Docs nennen „o4-mini“. ([GitHub](https://github.com/openai/codex), [OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started))

**Open-Source-Modelle (Ollama):**

* Schnellstart: codex --oss -m gpt-oss:20b (oder :120b).
* Basis-URL kann per CODEX\_OSS\_BASE\_URL/CODEX\_OSS\_PORT oder in config.toml gesetzt werden. ([GitHub](https://github.com/openai/codex))

**CI/Headless (nicht interaktiv):**

# GitHub Actions Beispiel

- name: Update changelog via Codex

run: |

npm install -g @openai/codex

export OPENAI\_API\_KEY="${{ secrets.OPENAI\_KEY }}"

codex exec --full-auto "update CHANGELOG for next release"

([GitHub](https://github.com/openai/codex))

**Zero-Data-Retention (ZDR) Orgs:**

codex --config disable\_response\_storage=true

# oder in ~/.codex/config.toml:

# disable\_response\_storage = true

([GitHub](https://github.com/openai/codex))

**Logging & Diagnose:**

* Datei-Log TUI: ~/.codex/log/codex-tui.log → tail -F ~/.codex/log/codex-tui.log
* RUST\_LOG für Verbosität (CLI in Rust). ([GitHub](https://github.com/openai/codex))

**Windows-Hinweise:**

* Offiziell **WSL2** nutzen; Sandbox auf reinem Windows nicht unterstützt. ([GitHub](https://github.com/openai/codex))

**2) Befehle, Optionen & Speicherpfade (Praxis-Referenz)**

**Top-Level Befehle:**

* codex – interaktive TUI (optional mit Start-Prompt: codex "…") ([GitHub](https://github.com/openai/codex))
* codex exec "…" – **nicht-interaktiv** (Automation/CI) ([GitHub](https://github.com/openai/codex))
* codex --login / codex --free – Login/Promo-Credits Flow ([OpenAI Help Center](https://help.openai.com/en/articles/11381614-codex-cli-and-sign-in-with-chatgpt))
* codex mcp – **experimentell** als MCP-Server laufen lassen ([GitHub](https://github.com/openai/codex))
* codex --upgrade / codex --version / codex help – Update/Version/Hilfe ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started))

**Wichtige Optionen (Stand heute):**

* -m, --model <NAME> – Modellwahl (Responses-API Modelle; Default variiert je Release) ([GitHub](https://github.com/openai/codex))
* --ask-for-approval <never|on-request|…> – Rückfrage-Policy (neu: on-request) ([GitHub](https://github.com/openai/codex/releases?utm_source=chatgpt.com))
* --sandbox <read-only|workspace-write|danger-full-access> – Sandbox-Modus ([GitHub](https://github.com/openai/codex))
* --auto-edit / --full-auto / **Default „Suggest“** – Kurzschalter für Arbeitsmodus ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started))
* --oss – OSS-Provider (Ollama); Default-BaseURL http://localhost:11434/v1 ([GitHub](https://github.com/openai/codex))
* --config key=value – ad-hoc Konfig-Override (auch für disable\_response\_storage=true) ([GitHub](https://github.com/openai/codex))
* --dangerously-bypass-approvals-and-sandbox – nur in isolierten Containern nutzen (Security-Bypass) ([GitHub](https://github.com/openai/codex))

Hinweis: Die vollständige Liste ist **versioniert und im Fluss**. Für die jeweils aktuelle Ausgabe codex help bzw. Releases konsultieren. ([GitHub](https://github.com/openai/codex/releases?utm_source=chatgpt.com))

**Konfiguration / Speicherorte:**

* ~/.codex/config.toml – **Hauptkonfiguration** (Profile, Sandbox, Provider, Projekte) ([GitHub](https://github.com/openai/codex))
* ~/.codex/AGENTS.md, <repo>/AGENTS.md, <cwd>/AGENTS.md – **Guidance** (Merge-Reihenfolge top-down) ([GitHub](https://github.com/openai/codex))
* ~/.codex/auth.json – **Anmeldedaten** (Migration/Sign-in Flow nennt Pfad explizit; Windows Host: C:\Users\<Name>\.codex\auth.json) ([GitHub](https://github.com/openai/codex))
* ~/.codex/log/codex-tui.log – **Logs** (TUI) ([GitHub](https://github.com/openai/codex))

**Azure/Open-Source Nutzung (Auszug):**

* Azure erfordert passenden **Provider** und **Deployment-Namen** als model. ([GitHub](https://github.com/openai/codex/issues/2024?utm_source=chatgpt.com))
* OSS via Ollama: Base-URL http://localhost:11434/v1 + --oss / model\_providers.oss. ([GitHub](https://github.com/openai/codex))

**MCP-Server einbinden (Clients/Tools):** ~/.codex/config.toml → [mcp\_servers.<name>] (command/args/env). ([GitHub](https://github.com/openai/codex))

**3) AGENTS.md – Rollen & Leitplanken (Python & Flutter)**

**Zweck:** Codex mit klaren Rollen, Befehls-Leitplanken und Qualitätsstandards versorgen. Datei ins **Repo-Root** legen (global zusätzlich ~/.codex/AGENTS.md möglich; Merge-Reihenfolge beachten). ([GitHub](https://github.com/openai/codex))

**Wichtig:**  
– Kein Remote-Git, keine Netzwerkaktionen ohne explizite Freigabe.  
– Python: ruff (Lint), black (Format), pytest (+pytest-cov), optional mypy.  
– Flutter/Dart: flutter analyze, dart format, flutter test.  
– Android: ./gradlew :app:assembleDebug testDebugUnitTest.  
– iOS (CI-geeignet): flutter build ios --no-codesign (kein Signing notwendig).

# AGENTS.md — Projektleitfaden (Python & Flutter)

## Grundregeln

- Handle konservativ: Bevor du Dateien änderst, prüfe Build, Lints und Tests.

- Änderungen immer in kleinen Schritten mit erklärten Diffs.

- Keine Netzwerkzugriffe und keine Git-Operationen (Push/PR) ohne explizite Anweisung.

- Respektiere .editorconfig, pyproject.toml und pubspec.yaml.

- Bevorzugte Sprachen: Python 3.12+, Dart/Flutter (stable channel).

## Build-/Test-Kommandos

### Python

- Setup/Env: `python -m venv .venv && source .venv/bin/activate`

- Abhängigkeiten: `pip install -U pip && pip install -r requirements.txt`

- Lint: `ruff check .`

- Format: `black .`

- Typprüfung (optional): `mypy .`

- Tests: `pytest -q --maxfail=1 --disable-warnings`

- Coverage: `pytest --cov --cov-report=term-missing`

### Flutter (Android/iOS)

- SDK prüfen: `flutter --version`

- Abhängigkeiten: `flutter pub get`

- Analyse: `flutter analyze`

- Format: `dart format .`

- Tests: `flutter test --machine`

- Android Build: `(cd android && ./gradlew :app:assembleDebug testDebugUnitTest)`

- iOS Build (CI-tauglich): `flutter build ios --no-codesign`

---

## Core CLI Agents

### 1) Executor

\*\*Ziel:\*\* Aufgaben in korrekt sequenzierten Schritten ausführen.

\*\*Darf:\*\* lokale Shell-Kommandos ausführen (Sandbox), Dateien lesen/schreiben.

\*\*Vorgehen:\*\*

1. Status erfassen (`/status`), dann Build/Lint/Test.

2. Kleine, reversible Patches anwenden; nach jedem Patch Tests laufen lassen.

3. Keine Netzwerke, keine Git-Remotes.

### 2) Parser

\*\*Ziel:\*\* Informationen extrahieren (Logs, Fehlermeldungen, API-Signaturen).

\*\*Darf:\*\* Lesen/Kommentieren, keine Schreibbefehle.

\*\*Output:\*\* strukturierte Bullet-Points + nächste konkrete Schritte.

### 3) Validator

\*\*Ziel:\*\* Änderungen validieren.

\*\*Checks:\*\* `ruff`, `black --check`, `mypy` (falls aktiviert), `pytest`, `flutter test`, `flutter analyze`.

\*\*Ergebnis:\*\* „grün/rot“-Bericht mit Fundstellen und Fix-Vorschlägen.

### 4) Formatter

\*\*Ziel:\*\* Konsistentes Formatting erzwingen.

\*\*Kommandos:\*\* `black .`, `dart format .`

\*\*Stop-Kriterium:\*\* `black --check .` / `git diff --exit-code`.

### 5) Analyzer

\*\*Ziel:\*\* statische Codeanalyse & Architekturhinweise.

\*\*Werkzeuge:\*\* `ruff`, `mypy`, `flutter analyze`.

\*\*Output:\*\* Liste riskanter Stellen mit Priorität und kurzem Fix-Plan.

---

## Advanced CLI Agents

### 6) Optimizer

\*\*Ziel:\*\* Performance und Speicherprofil verbessern.

\*\*Python:\*\* Hotspots identifizieren; Vorschläge (Algorithmus, Datenstruktur, I/O).

\*\*Flutter:\*\* Build-Zeit, Render-Overdraw, Rebuilds reduzieren.

### 7) Debugger

\*\*Ziel:\*\* Fehler reproduzieren und eng eingrenzen.

\*\*Vorgehen:\*\* minimaler Repro, Hypothesenliste, Fix ausprobieren, Tests.

### 8) Profiler

\*\*Ziel:\*\* Messbar machen.

\*\*Python:\*\* `pytest -k <kritisch>` plus Timing; Empfehlung für `cProfile`/`py-spy`.

\*\*Flutter:\*\* Hinweise zu `flutter run --profile`, DevTools (nur dokumentieren, nicht selbst starten).

### 9) Transformer (Refactorer)

\*\*Ziel:\*\* sichere Refactorings.

\*\*Leitlinien:\*\* kleine Schritte, API unverändert, Tests nach jedem Schritt.

### 10) Integrator

\*\*Ziel:\*\* Änderungen projektweit integrieren (Imports, Modulgrenzen, CI-Skripte).

\*\*Grenzen:\*\* Keine Remote-Git-Aktionen; nur lokale Anpassungen.

---

## Specialized CLI Agents

### 11) API Client

\*\*Ziel:\*\* stabile API-Schichten generieren/aktualisieren.

\*\*Python:\*\* HTTP-Clients mit Retries/Timeouts; Typen/Dataklassen.

\*\*Flutter:\*\* `dio`/`http`-Clients, JSON-Serialisierung.

### 12) File Processor

\*\*Ziel:\*\* Bulk-Operationen auf Dateien (Umbenennen, Header, Lizenzen).

\*\*Kommandos:\*\* sicheres `git mv` im Workspace; Regex-Audits mit Report.

### 13) Data Mapper

\*\*Ziel:\*\* DTO↔Domain-Mapping konsistent halten; Migrationshinweise.

### 14) Test Runner

\*\*Ziel:\*\* Teststrategie; Lücken finden & schließen.

\*\*Kommandos:\*\* `pytest -q`, `flutter test --coverage`.

\*\*Output:\*\* zu ergänzende Testfälle (Priorisierung: kritische Pfade).

### 15) Doc Generator

\*\*Ziel:\*\* technische Doku/CHANGELOG/README aktualisieren.

\*\*Quellen:\*\* Code-Diffs & Commits; erzeugt docs in `/docs` oder README-Abschnitte.

---

## Qualitätsbarrieren (für alle Agenten)

- Kein Netz, keine geheimen Keys in Klartext, keine externen Telemetrie-Pakete.

- Wenn Tests/Analyse rot → sofort stoppen, Bericht statt Folgeschritte.

- Jede Änderung muss reproduzierbar sein (Befehle + Gründe).

**Ergänzungen für deine Umgebung**

* **Projekt-spezifische Presets**: Du kannst für „Trusted“-Projekte trust\_level in die TOML packen (siehe Issue-Beispiel), um das Standardverhalten je Pfad anzupassen. ([GitHub](https://github.com/openai/codex/issues/2182?utm_source=chatgpt.com))
* **Ollama/Local**: Für Air-gapped-Betrieb ~/.codex/config.toml pflegen statt viele --config-Flags – etablierter Tipp in der Community. ([Reddit](https://www.reddit.com/r/ollama/comments/1lucq0b/codexollama_airgapped/?utm_source=chatgpt.com))

**Kurzfazit zu deiner Eingangsfrage**

* **Global unter WSL installieren** (npm -g @openai/codex) ist der richtige Weg. Pro-Projekt gibt es **keine** gesonderte Installation, nur **Arbeitsordner wählen** und optional AGENTS.md/config.toml pflegen. ([GitHub](https://github.com/openai/codex), [OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started))

Möchtest du zusätzlich eine minimal harte ~/.codex/config.toml für **zwei Profile** (Full-Auto vs. Read-Only) und ein drittes Profil „Ollama-Local“ von mir, liefere ich dir die TOML-Snippets direkt einsetzbar.  
  
  
  
Kurzantwort: **Ja – eigene Office-Agenten lohnen sich**, aber **nicht als “WYSIWYG-Layouter”**. Codex CLI ist ein **Coding-Agent**: Er liest/schreibt Dateien und führt Shell-/Python-Kommandos aus. Für Office-Aufgaben soll er **Tools und Bibliotheken orchestrieren** (z. B. python-docx/openpyxl/python-pptx, Pandoc, LibreOffice-Headless) – dann funktioniert es stabil. Für pixelgenaues Corporate-Layout taugt ein LLM allein nicht. ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started?utm_source=chatgpt.com), [OpenAI](https://openai.com/index/introducing-codex/?utm_source=chatgpt.com))

**Wann sinnvoll?**

* **Berichte/Protokolle automatisch erzeugen** (DOCX), **Tabellen** zusammenführen/prüfen (XLSX), **Slides** generieren (PPTX). Das geht sauber mit **python-docx**, **openpyxl**, **python-pptx**. ([python-docx.readthedocs.io](https://python-docx.readthedocs.io/?utm_source=chatgpt.com), [openpyxl.readthedocs.io](https://openpyxl.readthedocs.io/?utm_source=chatgpt.com), [python-pptx.readthedocs.io](https://python-pptx.readthedocs.io/?utm_source=chatgpt.com))
* **Mass-Konvertierungen & Text-Workflow**: DOCX⇄MD/HTML via **Pandoc**; PDF-Erzeugung via **LibreOffice –-headless**. Damit bleiben Diffs/Reviews textbasiert. ([Pandoc](https://pandoc.org/MANUAL.html?utm_source=chatgpt.com" \o "Pandoc User's Guide), [anarc.at](https://anarc.at/blog/2019-10-08-libreoffice-pdf-batch/?utm_source=chatgpt.com))

**Wann nicht?**

* **Layout-kritische Vorlagen** (komplizierte Seitenumbrüche, Feldfunktionen, SmartArt). Hier: Template + gezielte Platzhalter füllen (z. B. **docxtpl**) und nur minimal bearbeiten. ([docxtpl.readthedocs.io](https://docxtpl.readthedocs.io/?utm_source=chatgpt.com))

**Taugt Codex CLI dafür?**

Ja – **als Orchestrator**. Codex kann Dateien lesen/ändern und **Kommandos** ausführen; Autonomie/Stromschiene regelst du über *Suggest/Auto-Edit/Full-Auto* und *Sandbox*-Modi. Damit lassen sich die oben genannten Tools zuverlässig steuern. **Es ist jedoch kein Office-Editor**, sondern ein Automationslayer. ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started?utm_source=chatgpt.com), [GitHub](https://github.com/openai/codex?utm_source=chatgpt.com))

**Minimal-Setup (WSL)**

* Python-Libs: pip install python-docx openpyxl python-pptx ([PyPI](https://pypi.org/project/python-docx/?utm_source=chatgpt.com" \o "python-docx), [openpyxl.readthedocs.io](https://openpyxl.readthedocs.io/?utm_source=chatgpt.com))
* Konvertierung: **Pandoc** (DOCX↔MD) und **LibreOffice –-headless** (DOCX/XLSX/PPTX→PDF). ([Pandoc](https://pandoc.org/MANUAL.html?utm_source=chatgpt.com" \o "Pandoc User's Guide), [anarc.at](https://anarc.at/blog/2019-10-08-libreoffice-pdf-batch/?utm_source=chatgpt.com))

**Agents.md – Office-Rollen (ergänzend zu deinen Coding-Agenten)**

Lege dies **im Repo-Root** ab; Codex lädt/mischt globale und repo-lokale Regeln. ([GitHub](https://github.com/openai/codex?utm_source=chatgpt.com))

## Office Agents (DOCX/XLSX/PPTX)

### DocEditor

Ziel: Berichte/Schriftsätze aus Templates generieren und Inhalte aktualisieren.

Werkzeuge: python-docx (keine direkten Binär-Patches), optional docxtpl.

Vorgehen:

1) Vorlage prüfen, Platzhalterliste extrahieren.

2) Inhalte einfügen (Überschriften, Tabellen, Nummerierungen), keine Styles erfinden.

3) Validierung: Öffnen & Textprüfung, anschließend optional Pandoc→Markdown zur Review.

Grenzen: Kein manuelles Layout-Tuning, keine SmartArt/Track-Changes.

### SheetAgent

Ziel: Daten konsolidieren, Formeln/Funktionen setzen, CSV↔XLSX.

Werkzeuge: openpyxl.

Vorgehen: Nur bestehende Tabellenblätter anfassen, Datentypen erhalten, Formeln testweise evaluieren.

Validierung: Rechenproben + Export als CSV zur Sichtprüfung.

### SlideAgent

Ziel: Foliensätze aus Story/Outline erzeugen oder aktualisieren.

Werkzeuge: python-pptx.

Vorgehen: Titel+Bullet-Struktur, simple Tabellen/Diagramm-Stub; keine komplexen Layout-Experimente.

Validierung: Zählung von Folien/Platzhaltern, Stichproben-Screenshots nur wenn Tool vorhanden.

### ConvertAgent

Ziel: Sichere Konvertierungen & PDF-Erzeugung.

Werkzeuge:

- Pandoc: `pandoc in.docx -t markdown -o out.md`

- LibreOffice headless: `soffice --headless --convert-to pdf --outdir out in.docx`

Vorgehen: Konvertieren, Ergebnisgröße/Seiten prüfen, keine Layoutgarantien versprechen.

### OfficeValidator

Ziel: Qualitätsbarriere.

Checks:

- DOCX: Pflichtabschnitte/Anhänge vorhanden? Stichwortsuche.

- XLSX: Erwartete Sheets/Spalten vorhanden? Stichprobenformeln.

- PPTX: Agenda/Schlussfolie vorhanden? Keine leeren Platzhalter.

Abbruch bei Fehlern, Report statt Folgeschritte.

### Sicherheitsregeln (alle Office-Agenten)

- Keine externen Downloads/Netzwerkzugriffe.

- Nur im Workspace arbeiten; keine Binär-Diffs, immer Bibliothek/Tool nutzen.

- Bei Unklarheiten: erst Markdown-Entwurf erzeugen, dann ins Zielformat transformieren.

**Praxisfazit**

* **Sinnvoll**: Ja, wenn deine Office-Agenten **Bibliotheken/Converter steuern** und klar validieren.
* **Codex CLI** ist dafür **gut genug** – als Automatisierer mit klaren Leitplanken; er ersetzt **kein** visuelles Office. Für Python/Flutter-Projekte passen diese Agenten sauber in deinen bestehenden Workflow. ([OpenAI Help Center](https://help.openai.com/en/articles/11096431-openai-codex-cli-getting-started?utm_source=chatgpt.com), [GitHub](https://github.com/openai/codex?utm_source=chatgpt.com))

Wenn du willst, liefere ich dir noch ein kurzes ~/.codex/config.toml-Profil „office\_safe“ (read-only + On-Request) und „office\_batch“ (workspace-write + Auto-Edit) – direkt einsetzbar.