OpenAI Codex – Die komplette Anleitung

Terminal-Agent, Cloud, IDE-Integrationen und der Umstieg von Claude Code

Stand: Juli 2026

Christian Drapatz

OpenAI Codex CLI · Cloud · IDE-Erweiterungen · Desktop-App

Diese Anleitung stützt sich auf offizielle OpenAI-/Codex-Dokumentation (developers.openai.com/codex, learn.chatgpt.com/docs, github.com/openai/codex, help.openai.com), auf offizielle Blogposts sowie, wo keine offizielle Quelle auffindbar war, auf Community-Berichte, Drittanbieter-Blogs und Presseartikel. Codex entwickelt sich derzeit sehr schnell (mehrere Modell- und CLI-Generationen allein zwischen Ende 2025 und Mitte 2026), einzelne Zahlen und Namen können daher beim Lesen bereits wieder veraltet sein. Eine Übersicht der am schwächsten belegten Einzelpunkte steht am Ende unter „Hinweis zur Recherchequalität" (Kapitel 19).

Inhaltsverzeichnis

Rechtsklick auf das Feld → Felder aktualisieren, um das Verzeichnis zu befüllen.

1. Was ist Codex?

„Codex" ist heute kein einzelnes Werkzeug mehr, sondern ein Oberbegriff für vier zusammengehörige Produkte von OpenAI:

- **Codex CLI** – ein Open-Source-Terminal-Agent, der lokal auf dem eigenen Rechner läuft (github.com/openai/codex). Er liest, schreibt und führt Code in einer Sandbox aus, direkt im Projektverzeichnis.
- **Codex Cloud / Codex Web** (chatgpt.com/codex) – die browserbasierte Variante. Aufgaben laufen in isolierten Cloud-Sandboxen, mehrere Tasks parallel, mit GitHub-Integration bis hin zur automatischen Pull-Request-Erstellung.
- **IDE-Erweiterungen** – eine VS-Code-Extension (funktioniert auch in VS-Code-Forks wie Cursor oder Windsurf) sowie eine native Integration in JetBrains-IDEs (IntelliJ, PyCharm, WebStorm, Rider u. a.).
- **Codex-Desktop-App** – eine eigenständige App für macOS und Windows für Projektarbeit, Markdown-Editing und lang laufende Aufgaben.

Alle vier teilen sich dieselben Modelle und (je nach Plan) dasselbe Nutzungskontingent, unterscheiden sich aber deutlich darin, ob man synchron im Terminal wartet oder asynchron im Hintergrund arbeiten lässt.

Wichtig zur Begriffsklärung: „Codex" gab es schon einmal – das ursprüngliche OpenAI-Codex-Modell aus dem August 2021, ein von GPT-3 abgeleitetes, auf Milliarden Zeilen öffentlichen Codes feingetuntes Modell. Es war der Motor der ersten GitHub-Copilot-Version, wurde aber am 23. März 2023 als eigenständiges Modell/API abgekündigt und in GPT-3.5/GPT-4 aufgehoben. Zwei Jahre lang existierte der Name „Codex" praktisch nicht mehr. Am 16. April 2025 kehrte er zurück – als komplett neues Produkt: der Open-Source-Terminal-Agent Codex CLI. Das heutige Codex ist also ein agentisches Coding-Werkzeug (liest/schreibt/führt Code aus, plant mehrschrittige Aufgaben), keine reine Code-Vervollständigung wie das alte Modell.

Kurz danach, im Mai/Juni 2025, kam die Cloud-Variante als asynchroner „Software-Engineering-Agent" in ChatGPT dazu, angetrieben vom Modell „codex-1" (einer auf Coding optimierten Variante von o3). Seither folgt eine dichte Kette codex-spezialisierter Modelle (GPT-5-Codex, GPT-5.1-Codex, GPT-5.2-Codex, GPT-5.3-Codex …), die CLI wurde zwischenzeitlich vollständig von TypeScript nach Rust umgeschrieben, und die IDE- sowie Desktop-Anbindungen kamen sukzessive dazu.

2. Wem gehört Codex?

Codex wird von **OpenAI** entwickelt und betrieben. OpenAI wurde im Dezember 2015 als gemeinnützige KI-Forschungsorganisation gegründet, mit dem erklärten Ziel, dass „AGI der gesamten Menschheit nützen soll". 2019 entstand eine „capped-profit"-Tochtergesellschaft unter Kontrolle der gemeinnützigen Mutterorganisation, mit auf das 100-fache gedeckelten Investorenrenditen.

Am 28. Oktober 2025 schloss OpenAI eine grundlegende Rekapitalisierung ab: Die neu geschaffene **OpenAI Foundation** (weiterhin Non-Profit) kontrolliert seither die **OpenAI Group PBC** (eine Public Benefit Corporation) und hält daran einen wirtschaftlichen Anteil von rund 26 %, bei voller Kontrolle über die Governance. **Microsoft** ist seit seiner ersten Investition 2019 (1 Mrd. USD) der größte strategische Investor und hält inzwischen rund 27 % des wirtschaftlichen Anteils. Anfang 2026 folgte eine weitere Rekordfinanzierungsrunde über rund 110 Mrd. USD (u. a. mit Amazon, Nvidia und SoftBank als Investoren) bei einer Bewertung von etwa 730 Mrd. USD.

Der Quellcode der Codex CLI selbst ist unter der Apache-2.0-Lizenz offen (github.com/openai/codex) – die dahinterliegenden Modelle sind es nicht; sie laufen ausschließlich über OpenAIs eigene Infrastruktur (ChatGPT-Konto oder API-Key).

3. Architektur: CLI, Cloud, IDE-Erweiterungen, Desktop-App

Der offizielle Quickstart-Guide nennt ausdrücklich vier gleichberechtigte Zugangswege zu Codex:

Zugang	Wo läuft die Aufgabe?	Typischer Einsatz
Codex CLI	Lokal, im eigenen Terminal	Interaktives Pair-Programming im Repo, volle Dateisystemkontrolle
Codex Cloud / Web (chatgpt.com/codex)	In einer isolierten Cloud-Sandbox von OpenAI	Mehrere Aufgaben parallel im Hintergrund, PRs aus Issues/Slack/Linear
IDE-Erweiterung (VS Code, JetBrains)	Wahlweise lokal oder delegiert an die Cloud	Arbeiten im gewohnten Editor, ohne das Terminal zu wechseln
Desktop-App (macOS, Windows)	Lokal bzw. gegen die Cloud	Eigenständige App für Projektarbeit, Markdown-Editing, PR-Reviews

Alle vier Wege nutzen dieselben Modelle und – je nach Abo – dasselbe Nutzungskontingent. Für dieses Tutorial liegt der Schwerpunkt auf der **Codex CLI**, da sie am ehesten mit Claude Code vergleichbar ist; Kapitel 9 und 11 gehen gesondert auf Cloud und IDE-Integrationen ein.

4. Installation

4.1 Systemanforderungen

Offiziell voll unterstützt sind **macOS** (Apple Silicon und Intel) und **Linux** (x86_64 und arm64). **Windows** gilt weiterhin als experimentell: Es gibt zwar seit Anfang 2026 eine native Windows-Installation mit AppContainer-basierter Sandbox, für den produktiven Einsatz wird aber weiterhin empfohlen, Codex innerhalb eines **WSL2**-Workspace laufen zu lassen – dort greift dieselbe Landlock/seccomp-Sandbox wie unter nativem Linux, also die Umgebung, in der die Modelle primär trainiert wurden.

4.2 Schritt-für-Schritt-Installation

Am gängigsten ist die Installation über npm:

```
npm install -g @openai/codex
```

Achtung bei der Schreibweise: `npm install -g codex` (ohne den Scope `@openai/`) installiert ein völlig unabhängiges, seit 2012 existierendes Paket gleichen Namens – nicht OpenAIs Codex CLI.

Alternativ über Homebrew:

```
brew install --cask codex
```

Oder über den Shell-Installer, direkt von OpenAI bereitgestellt:

```
# macOS/Linux
curl -fsSL https://chatgpt.com/codex/install.sh | sh

# Windows (PowerShell)
powershell -ExecutionPolicy ByPass -c "irm
https://chatgpt.com/codex/install.ps1 | iex"
```

Wer keinen Paketmanager nutzen möchte, kann auch die plattformspezifischen Binaries direkt von den GitHub-Releases herunterladen (github.com/openai/codex/releases).

Test der Installation:

```
codex --version
codex doctor   # Diagnose: prüft Installation, Auth-Status, Sandbox-Fähigkeit
```

Codex starten: Im Projektverzeichnis genügt der Aufruf ohne weitere Argumente, um die interaktive TUI zu öffnen:

Codex TUI nach dem ersten Start

```
cd mein-projekt
codex
```

Alternativ lässt sich direkt eine erste Aufgabe als Prompt übergeben, mit der die Session sofort startet:

```
codex "Erkläre mir die Struktur dieses Projekts"
```

Für nicht-interaktive Aufrufe (z. B. in Skripten oder CI/CD) gibt es den Modus `codex exec`, der ohne TUI arbeitet und direkt das Ergebnis ausgibt (siehe 7.3):

```
codex exec "Führe die Tests aus und fasse Fehlschläge zusammen"
```

Beim allerersten Start ohne vorherige Anmeldung führt Codex automatisch durch den Login-Vorgang aus Abschnitt 4.3.

4.3 Anmeldung (`codex login`)

Codex kennt zwei Authentifizierungswege:

```
codex login
```

startet standardmäßig **„Sign in with ChatGPT"** (OAuth im Browser) – dabei wird das Nutzungskontingent des eigenen ChatGPT-Plans (Plus, Pro, Business, Edu oder Enterprise) verwendet, ohne dass zusätzlich Token nach API-Preisen abgerechnet werden.

Alternativ kann man sich mit einem **API-Key** von platform.openai.com anmelden – dann greift die nutzungsbasierte API-Abrechnung (siehe Kapitel 10). Je nach gewähltem Weg können einzelne Features unterschiedlich verfügbar sein. `codex login` unterstützt laut CLI-Referenz außerdem Device-Auth-Flows sowie das Einlesen eines Access-Tokens über stdin, was sich für CI/CD- oder Remote-Umgebungen eignet.

5. Ein Modell auswählen

Die aktuelle Modellgeneration (Stand Juli 2026) ist die **GPT-5.6-Familie**, die am 9. Juli 2026 vorgestellt wurde – mit einem neuen Namensschema: eine Zahl (5.6) markiert die Generation, ein Name (Sol / Terra / Luna) das Fähigkeits- und Preistier, sodass beide unabhängig voneinander weiterentwickelt werden können:

- **Sol** – Flaggschiff für komplexes Coding, Computer-Use, Research und sicherheitsnahe Aufgaben.
- **Terra** – ausgewogenes Modell, günstiger, als „natürlicher Startpunkt" für die tägliche Arbeit gedacht.
- **Luna** – das schnellste und günstigste Modell, für hochvolumige, gut vorhersehbare Aufgaben.

Davor lag die Codex-spezialisierte Modellreihe GPT-5-Codex → GPT-5.1-Codex → GPT-5.2-Codex (11. Dezember 2025) → GPT-5.3-Codex, teils mit einer separaten „Spark"-Preview-Variante für besonders schnelle Iteration (nur ChatGPT Pro). Zusätzlich lässt sich der Reasoning-Aufwand einstellen (von niedrigen Stufen bis „Max" für tiefe Einzelaufgaben-Analyse und „Ultra" für parallele Sub-Agent-Delegation).

Das Modell lässt sich an mehreren Stellen wählen:

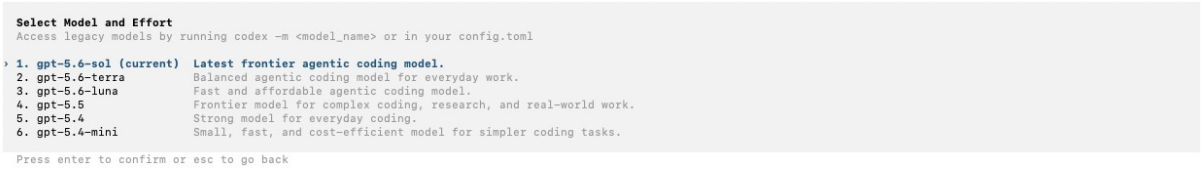
```
codex --model gpt-5.6      # einmalig per CLI-Flag
# ~/.codex/config.toml
model = "gpt-5.6"
model_reasoning_effort = "high"
```

oder direkt in der Desktop-App bzw. im Web-UI über ein Dropdown.

Innerhalb einer laufenden CLI-Session lässt sich das Modell außerdem jederzeit ohne Neustart wechseln, per Slash-Command:

```
/model
```

Der Befehl öffnet eine interaktive Auswahl der verfügbaren Modelle (inkl. Reasoning-Stufe), sodass man z. B. für eine einzelne aufwendige Aufgabe kurz auf Sol mit hohem Reasoning-Aufwand wechseln und danach wieder zu Terra zurückkehren kann, ohne die Session zu verlassen.



Übersicht der verfügbaren Codex-Modelle

Hinweis: Modellnamen und -generationen ändern sich bei Codex derzeit sehr schnell – teils mehrfach pro Quartal. Vor dem produktiven Einsatz lohnt sich ein Blick auf `codex --help` bzw. die aktuelle Modellliste unter learn.chatgpt.com/docs/models, um zu prüfen, welche Namen aktuell tatsächlich gültig sind.

6. Codex einrichten: der erste Workspace

1. In das gewünschte Projektverzeichnis wechseln: `cd mein-projekt`
2. `codex` ohne weitere Argumente starten – das öffnet die interaktive Terminal-UI (TUI).
3. Beim ersten Start in einem neuen Verzeichnis fragt Codex, ob das Projekt als „vertrauenswürdig" (trusted) eingestuft werden soll. Das beeinflusst, ob eine lokale `.codex/config.toml` automatisch geladen wird und welche Sandbox-/Approval-Defaults gelten.
4. Einen ersten Prompt eingeben, z. B. „Erkläre mir die Struktur dieses Repositories" – Codex liest dafür zunächst read-only die Projektdateien.
5. Für einen konkreten Auftrag („Füge einen Unit-Test für X hinzu") schlägt Codex einen Plan vor und fragt vor Schreibzugriffen oder Kommandoausführungen um Freigabe – abhängig vom gewählten Approval-Modus (siehe 7.2).

Alternativ lässt sich Codex direkt mit einem Prompt starten:

```
codex "Analysiere die Testabdeckung in diesem Repository und schlage fehlende
Tests vor"
```

7. Arbeiten mit der CLI

7.1 Wichtige Befehle

Befehl	Zweck
`codex`	interaktive Terminal-UI starten
`codex exec` (Alias `codex e`)	nicht-interaktiver Modus, siehe 7.3
`codex resume`	vorherige Session fortsetzen (per ID oder zuletzt genutzte)
`codex login`	Anmeldung (ChatGPT-OAuth oder API-Key)
`codex review`	nicht-interaktive Code-Analyse/Review
`codex doctor`	Diagnose von Installation, Auth, Sandbox
`codex mcp`	MCP-Server verwalten (hinzufügen, auflisten, entfernen, OAuth)
`codex app`	Desktop-App starten
`codex archive/unarchive/delete/fork SESSION`	Sessions verwalten

Wichtige globale Flags:

Flag	Werte	Zweck
`--model, -m`	z. B. `gpt-5.6`	Modell überschreiben
`--sandbox, -s`	`read-only`, `workspace-write`, `danger-full-access`	Sandbox-Level
`--ask-for-approval, -a`	`untrusted`, `on-request`, `never`	Freigabeverhalten
`--cd, -C`	Pfad	Arbeitsverzeichnis
`--profile, -p`	Name	zusätzliches Profil aus `config.toml`
`--config, -c`	`key=value`	Einzelwert-Override
`--search`	bool	Live-Web-Suche aktivieren

Innerhalb der TUI stehen zudem Slash-Commands zur Verfügung, u. a. `/skills`, `/memories`, `/theme`, `/vim`, `/hooks`, `/rename`, `/raw`.

7.2 Sandbox-Modi und Freigaben (Approval Policy)

Codex trennt zwei Dimensionen: **was** der Agent tun darf (Sandbox) und **wie oft** er dafür nachfragen muss (Approval Policy).

Sandbox-Modi (Nachfolger der älteren Begriffe „Suggest/Auto Edit/Full Auto" aus früheren CLI-Versionen):

- `read-only` – nur Lesezugriff im Projektverzeichnis
- `workspace-write` – Schreibzugriff innerhalb des Workspace erlaubt
- `danger-full-access` – keine Sandbox-Einschränkung mehr

Approval-Policy:

- `untrusted` – jede Aktion braucht eine explizite Rückfrage
- `on-request` – Rückfrage nur bei Unsicherheit
- `never` – läuft ohne Rückfragen durch

Technisch basiert die Sandbox auf macOS **Seatbelt** (`sandbox-exec`) sowie unter Linux auf einer Kombination aus **Landlock** (Dateisystem-Isolation) und **seccomp-BPF** (Syscall-Filter, der u. a. Netzwerkzugriffe blockiert). Standardmäßig gilt: kein ausgehender Netzwerkzugriff, Schreibrechte nur im freigegebenen Workspace. Unter Windows kommt eine AppContainer-basierte Sandbox zum Einsatz, die aber – Stand Anfang 2026 – noch als experimentell gilt.

7.3 Nicht-interaktiver Modus: `codex exec`

Für Skripte, CI-Pipelines oder Automatisierung eignet sich der Exec-Modus:

```
codex exec "Führe alle Tests aus und fasse Fehlschläge zusammen"
```

Nützliche Flags: `--json` (strukturierte JSONL-Events statt Klartext), `--output-last-message PATH` (letzte Antwort in eine Datei schreiben), `--ephemeral` (keine Session-Persistenz, für Einmal-Läufe in CI).

8. Konfiguration

8.1 `AGENTS.md`

`AGENTS.md` ist das Codex-Äquivalent zu Claude Codes `CLAUDE.md`: eine Markdown-Datei mit Projektkontext und Arbeitsanweisungen, die vor jeder Aufgabe geladen wird. Auflösungsreihenfolge:

1. **Globaler Scope** im Codex-Home-Verzeichnis (Standard `~/.codex`, überschreibbar via Umgebungsvariable `CODEX_HOME`): Zuerst wird `AGENTS.override.md` gesucht, sonst `AGENTS.md` – nur die erste nicht-leere Datei zählt.
2. **Projekt-Scope**: Ausgehend vom Projekt-/Git-Root abwärts bis zum aktuellen Arbeitsverzeichnis wird in jedem Verzeichnis nach `AGENTS.override.md`, dann `AGENTS.md`, dann konfigurierten Fallback-Dateinamen gesucht.

Alle gefundenen Dateien werden von oben (global) nach unten (näher am Arbeitsverzeichnis) **konkateniert** – bei Widersprüchen gewinnt die lokalere Datei. Leere Dateien werden ignoriert.

Praktisch heißt das: Ein Team kann projektweite Regeln in `AGENTS.md` im Repo-Root ablegen (Analogon zu `CLAUDE.md`) und zusätzlich unterverzeichnis-spezifische Regeln (z. B. in `frontend/AGENTS.md`) ergänzen.

8.2 `~/.codex/config.toml` und Profile

Konfigurationsquellen werden in dieser Rangfolge zusammengeführt (höchste Priorität zuerst):

1. CLI-Flags / `--config`-Overrides
2. Projekt-Config `.codex/config.toml` (nur bei „vertrauenswürdigen" Projekten geladen; das nächstgelegene Verzeichnis gewinnt)
3. Profildateien, aktiviert per `--profile name`
4. User-Config `~/.codex/config.toml`
5. System-Config `/etc/codex/config.toml` (Unix)
6. eingebaute Defaults

Beispiel:

```toml
model = "gpt-5.6"
approval_policy = "on-request"    # untrusted | on-request | never
sandbox_mode = "workspace-write"  # read-only | workspace-write | danger-full-access
model_reasoning_effort = "high"
personality = "pragmatic"         # oder "friendly", "none"
web_search = "cached"             # cached | live | disabled

[features]
memories = true
multi_agent = true
```

Organisationen können per zentraler `requirements.toml` Approval-Policy, Sandbox-Modus und MCP-Allowlists verbindlich vorgeben (Admin-Enforcement) – ähnlich den unternehmensweiten `managed-settings.json`-Regeln bei Claude Code.

8.3 MCP-Server

MCP-Server werden über `[mcp_servers.NAME]`-Tabellen in `config.toml` eingetragen. Für lokale (Stdio-)Server:

```toml
[mcp_servers.docs]
enabled = true
```

```
required = true
command = "docs-server"
args = ["--port", "4000"]
env = { "API_KEY" = "value" }
startup_timeout_sec = 10.0
tool_timeout_sec = 60.0
enabled_tools = ["search", "summarize"]
disabled_tools = ["slow-tool"]
```

Für Remote-Server über Streamable HTTP:

```
[mcp_servers.github]
enabled = true
url = "https://github-mcp.example.com/mcp"
bearer_token_env_var = "GITHUB_TOKEN"
http_headers = { "X-Custom" = "value" }
scopes = ["repo"]
```

Zusätzlich verwaltet man MCP-Server bequemer über das Subcommand `codex mcp` (Hinzufügen, Auflisten, Entfernen, OAuth-Flows direkt im Terminal).

Sicherheitshinweis: Im Dezember 2025 wurde eine kritische Schwachstelle (CVE-2025-61260, CVSS 9,8) bekannt, bei der Codex CLI MCP-Server-Einträge aus lokalen Projekt-Konfigurationen automatisch lud und ausführte, ohne interaktive Freigabe einzufordern – eine Command-Injection-Lücke. OpenAI hat den Fehler laut Berichten bereits vor der öffentlichen Bekanntgabe gepatcht. Wer ein fremdes Repository mit eigener `.codex/config.toml` öffnet, sollte trotzdem grundsätzlich prüfen, welche MCP-Server dort eingetragen sind, bevor man dem Projekt „vertrauenswürdig" freigibt.

8.4 Custom Prompts und Skills

Markdown-Dateien unter `~/.codex/prompts/` (nur Top-Level, keine Unterordner) lassen sich als Slash-Commands aufrufen – das älteste Anpassungsmittel der CLI, allerdings rein lokal und nicht über ein Repo teilbar.

Der Nachfolger sind **Skills**: Sie lassen sich über das Repository teilen und können von Codex – anders als Custom Prompts – auch implizit aufgerufen werden, wenn die Aufgabe dazu passt. `/skills` öffnet die Auswahl in der TUI. Das entspricht in etwa den Agent Skills, die auch Claude Code und Cursor inzwischen kennen.

8.5 Hooks (Stand: wenig dokumentiert)

Hooks existieren als Feature-Flag (`[features] hooks = true`) und als `/hooks`-Command in der TUI. Die genaue Konfigurationssyntax (welche Events es gibt, wie ein Hook-Skript eingebunden wird) war zum Zeitpunkt der Recherche nicht vollständig offiziell dokumentiert. Wer produktiv auf Hooks setzen will, sollte vor der Nutzung `codex --help` bzw. `/hooks` direkt in der aktuell installierten Version prüfen, statt sich auf ältere Blogposts zu verlassen.

9. Codex Cloud

Codex Cloud (chatgpt.com/codex) führt Aufgaben **asynchron und parallel** in isolierten Cloud-Sandboxen aus – jede Aufgabe bekommt vorab eine Kopie des Repositories geladen.

Ausgelöst werden Cloud-Tasks auf mehreren Wegen:

- direkt im Web-Interface,
- per `@codex`-Erwähnung in GitHub-Issues oder PR-Kommentaren,
- aus Linear-Issues heraus,
- aus Slack-Kanälen bzw. -Threads.

Für die **GitHub-Integration** wählt man aus, auf welche Repos Codex Zugriff hat; der Agent reagiert dann auf Erwähnungen, erstellt eigenständig Commits und Pull Requests zur Review und kann PRs auch selbst reviewen. Pro Repository lässt sich eine **Environment-Konfiguration** hinterlegen (Setup-Schritte, Abhängigkeiten, Umgebungsvariablen/Secrets), die für spätere Tasks wiederverwendet wird.

Der wesentliche Unterschied zur lokalen CLI: Die CLI arbeitet synchron im eigenen Terminal mit direktem Dateisystemzugriff und laufender Rückfrage; Codex Cloud arbeitet im Hintergrund weiter, auch wenn man den Browser schließt, und liefert das Ergebnis als Diff bzw. fertigen Pull Request zur Review. Das eignet sich für mehrere parallele, länger laufende Aufgaben, während die CLI näher am klassischen Pair-Programming bleibt.

10. Preise

Codex ist in mehreren ChatGPT-Plänen enthalten (Stand Juli 2026):

Plan	Preis	Hinweis
Free	0 $	minimales Kontingent
Go	8 $/Monat	eingeschränktes Kontingent für leichte Aufgaben
Plus	20 $/Monat	je nach Modell ca. 15–110 Codex-Nachrichten pro 5-Stunden-Fenster
Pro	100 $/Monat („5×“) bzw. 200 $/Monat („20×”)	deutlich höheres Kontingent, inkl. unbegrenzter Sprachnutzung bei „20×”
Business	25 $/Nutzer/Monat (bzw. 20 $ jährlich, mind. 2 Nutzer)	löste zum 2. April 2026 den alten Team-Plan ab
Enterprise	individuell	zentrale Verwaltung, eigene Datenschutzzusagen

Seit dem **2. April 2026** wird die Codex-Nutzung in ChatGPT-Plänen nicht mehr pro Nachricht, sondern nach **Token-/Credit-Verbrauch** abgerechnet, mit Kontingenten pro 5-Stunden-Fenster (modellabhängig – ein schnelleres/günstigeres Modell wie „Luna” erlaubt deutlich mehr Nachrichten im selben Fenster als das Flaggschiffmodell „Sol”).

Unabhängig vom ChatGPT-Abo lässt sich Codex auch mit einem **API-Key** nutzen und dann nach tatsächlichem Token-Verbrauch zu regulären API-Preisen abrechnen (Selbstzahler-Modell, ohne Abo-Kontingent). Die genauen Preise pro Modell und Million Token ändern sich mit jeder neuen Modellgeneration und sollten vor einer Kostenkalkulation direkt auf der offiziellen Pricing-Seite geprüft werden, da veröffentlichte Zahlen in Blogs und Aggregator-Seiten schnell veralten.

Praxishinweis: Mehrere Quellen beziffern die realen Durchschnittskosten für einen aktiv mit Codex arbeitenden Entwickler auf ungefähr 100–200 $ im Monat, abhängig vom gewählten Modell, der Zahl parallel laufender Cloud-Tasks und dem verwendeten Reasoning-Aufwand.

11. IDE-Integrationen und Desktop-App

11.1 VS-Code-Extension

VS-Code-Extension (Publisher „openai"): erscheint automatisch als Seitenleiste, funktioniert auch in VS-Code-Forks wie Cursor oder Windsurf. Anmeldung per ChatGPT-Konto oder API-Key; alle bezahlten ChatGPT-Pläne (Plus/Pro/Business/Edu/Enterprise) enthalten ein Codex-Nutzungskontingent. Aufgaben lassen sich wahlweise lokal oder an Codex Cloud delegieren; die Extension erstellt vor und nach jedem Task einen Git-Checkpoint, sodass sich Änderungen zurückrollen lassen.

11.2 JetBrains-Integration

Seit Januar 2026 nativ in IntelliJ IDEA, PyCharm, WebStorm, Rider u. a. ab Version 2025.3 – anders als bei VS Code kein eigenes Sidebar-Plugin, sondern direkt in die bestehende JetBrains-AI-Chat-Oberfläche eingebettet. Nutzbar mit JetBrains-AI-Abo, ChatGPT-Konto oder eigenem OpenAI-API-Key, alle im selben Chat-Fenster.

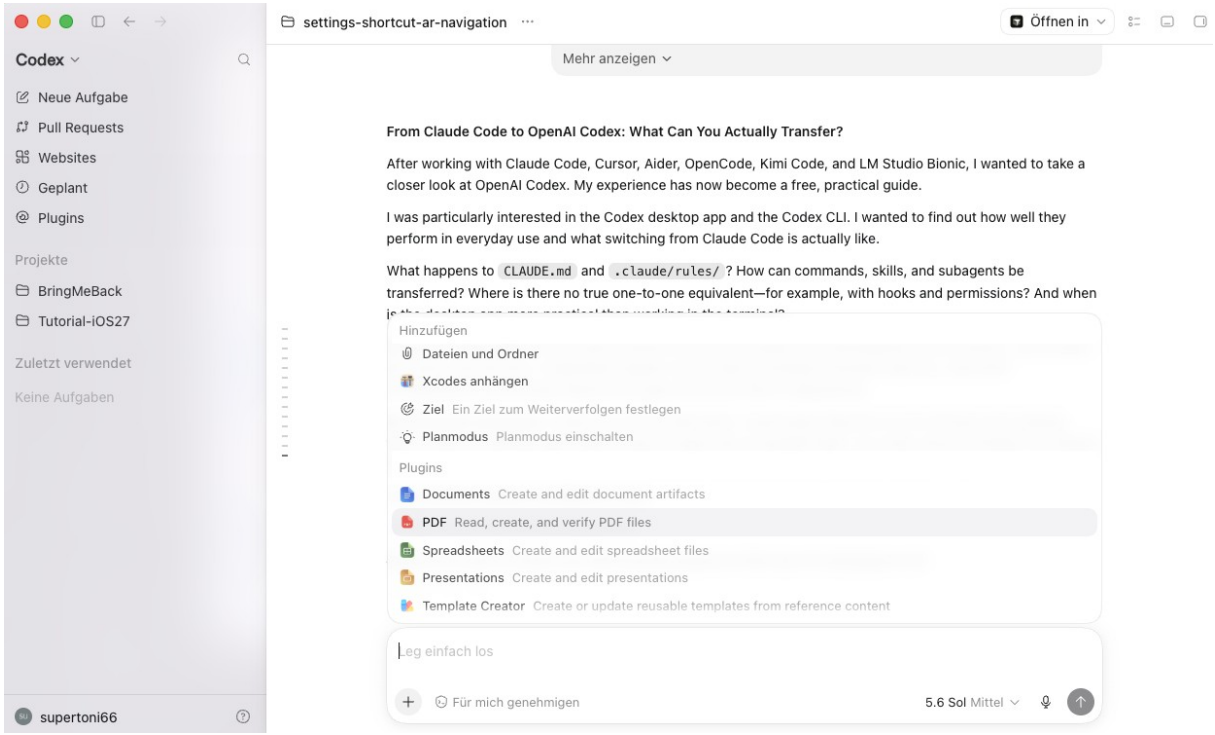
11.3 Die Codex-/ChatGPT-Desktop-App

Geschichte: Die Desktop-App startete am 2. Februar 2026 zunächst als eigenständige „Codex-App" für macOS – gedacht als „Kommandozentrum" für Agenten: mehrere Agenten gleichzeitig steuern und lang laufende, bis zu rund 30 Minuten autonom arbeitende Aufgaben überwachen. Windows-Support kam am 4. März 2026 dazu. Im Februar/März 2026 folgte unter dem Motto „Codex for (almost) everything" ein Funktionsschub: Computer Use, In-App-Browsing, Bildgenerierung, Memory und Plugins.

Wichtige Änderung zum 9. Juli 2026: Die bis dahin eigenständige Codex-App ist in der neuen, vereinheitlichten **ChatGPT-Desktop-App** aufgegangen – sie ist damit keine separate Anwendung mehr, sondern ein eigener Bereich neben „Chat" und „Work". Bestehende Codex-App-Nutzer erhalten die neue App per Update, Nutzer der alten ChatGPT-Desktop-App ein In-App-Upgrade-Angebot; Projekte, Chats und Einstellungen bleiben beim Umzug erhalten. Die alte, separate Codex-App-Version bleibt als „ChatGPT Classic" bestehen, bekommt aber nur noch Modell-Updates und Sicherheitspatches, keine neuen Agenten-Features mehr. Die vereinheitlichte App ist auf jedem ChatGPT-Plan verfügbar, einschließlich Free.

Installation: Download über die offizielle ChatGPT-Seite, verfügbar für macOS (Apple Silicon und Intel) und Windows. Anmeldung wie bei der CLI über zwei Wege: „Sign in with ChatGPT" (Browser-Login, Abrechnung über den ChatGPT-Plan) oder „Sign in with API Key" (Key von platform.openai.com, Abrechnung zu Standard-API-Preisen) – mit reinem API-Key sind einige Features eingeschränkt, die ChatGPT-Workspace- bzw. Cloud-Dienste voraussetzen.

Oberfläche und Funktionsumfang: Die App gliedert sich in drei Bereiche – **Chat** (normale Konversation), **Work** (Rechercheergebnisse und fertige Deliverables mit lokalem Dateizugriff) und **Codex** (Softwareentwicklung: lokale Dateien, Repositories, Terminals, Dev-Tools).



ChatGPT-Desktop-App mit den Bereichen Chat, Work und Codex

Eine linke Sidebar listet alle laufenden Chats/Threads, im zentralen Arbeitsbereich zeigt eine Side-by-side-Diff-Ansicht Agenten-Änderungen, ohne dass man in einen separaten Editor wechseln muss. Mit dem Juli-2026-Update kamen dazu:

- **Markdown-Editing mit Inline-Annotations** – Markdown und Code direkt in der App bearbeiten, Kommentare an bestimmten Stellen hinterlassen und Codex gezielt bitten, den markierten Abschnitt zu überarbeiten.
- **PR Chat** – GitHub-Pull-Requests direkt in der App reviewen: Codex zu einzelnen Änderungen befragen, Inline-Review-Feedback geben, vorgeschlagene Patches ansehen und annehmen/ablehnen/editieren, inklusive Multi-Repo-Unterstützung in einem Projekt.
- **Sites/Custom Domains** – von Codex/Work erzeugte, veröffentlichte Web-Artefakte („Sites") lassen sich an eigene Domains binden.
- **Steer** – ein laufender Agent lässt sich „on the fly" korrigieren, ohne ihn zu stoppen und neu zu starten; nach Nutzerberichten eines der nützlichsten Features, weil es den Unterbrechen-neu-anstoßen-Zyklus vermeidet. Zusätzlich verhindert die App, dass der Rechner während langer Agentenläufe in den Ruhezustand wechselt.
- **Parallele Aufgaben** – mehrere Projekte/Agenten lassen sich gleichzeitig verwalten, ähnlich wie in Codex Cloud.

Verhältnis zu CLI und Codex Cloud: Die App unterstützt beide Sandbox-Modelle und lässt pro Aufgabe wählen: eine **lokale Sandbox** wie bei der CLI (OS-erzwungen, Zugriff im Wesentlichen auf den aktuellen Workspace beschränkt, Rückfragen nach Approval-Policy) oder eine **Cloud-Sandbox** wie bei Codex Web (das Repository wird in eine isolierte Cloud-Umgebung geklont, eigenes Dateisystem, eigener Prozessraum, stark eingeschränkter Netzwerkzugriff). Sitzungsverlauf und Konfiguration werden dabei mit CLI und IDE-Extension geteilt, man kann also ohne Bruch zwischen den Werkzeugen wechseln; Cloud-Konversationen sind geräteübergreifend verfügbar, rein lokale Konversationen bleiben auf dem jeweiligen Rechner.

Konfiguration: Die App nutzt dieselbe Konfigurationsbasis wie die CLI – `~/.codex/config.toml` (Modell, `approval_policy`, `sandbox_mode`, Profile) und `AGENTS.md` für Projektkontext werden ebenso gelesen. Eigene App-Einstellungen (Modellwahl, Plugin-Verwaltung, Tastenkürzel) sind über

„Settings" erreichbar; eine detaillierte Liste aller Tastenkürzel war in den offiziellen Quellen zum Zeitpunkt der Recherche nicht vollständig dokumentiert.

11.4 Bekannte Einschränkungen der Desktop-App

Aus Nutzerberichten (Stand ca. März–Juli 2026) ergeben sich im Vergleich zur CLI einige wiederkehrende Kritikpunkte:

- **Kein manuelles `/compact`** – nur automatische Kontext-Kompression, die laut Berichten gelegentlich hängen bleibt, ohne manuellen Override.
- **Kein `--add-dir`-Äquivalent** – die App sieht nur das eine geöffnete Verzeichnis; für Monorepos oder projektübergreifende Shared Libraries fehlt die Möglichkeit, weitere Verzeichnisse hinzuzufügen.
- **Keine benutzerdefinierten Review-Kriterien**, die in der CLI konfigurierbar sind.
- **Weniger granulare Kontrolle** über Sandbox-Details und Konfigurationspfade als in der CLI.
- **Vereinzelt instabile Git-Diffs** und abgelehnte Änderungen, laut mindestens einem Nutzerbericht ein Grund, für bestimmte Aufgaben zur CLI zurückzuwechseln.
- **Performance**: gelegentliches UI-Lag bei langen Chats mit vielen Bildanhängen.

Eine in Community-Beiträgen häufig genannte Praxis: die CLI parallel installiert lassen und bei hängenden App-Sessions kurz dorthin wechseln (z. B. für `/compact`), bevor man in der App weiterarbeitet.

12. Vergleich mit Claude Code

Codex CLI und Claude Code sind sich architektonisch sehr ähnlich: beides reine Terminal-Agenten (keine eigene IDE), die neben dem bestehenden Editor laufen, projektweite Kontextdateien lesen (`AGENTS.md` bzw. `CLAUDE.md`), über Sandbox-/Approval-Mechanismen Datei- und Kommandozugriff steuern und MCP-Server unterstützen. Die wichtigsten Unterschiede:

Aspekt	Codex CLI	Claude Code
Anbieter/Modelle	OpenAI, ausschließlich eigene GPT-5.x-Codex-Modelle	Anthropic, ausschließlich eigene Claude-Modelle
Kontextdatei	`AGENTS.md` (+ `AGENTS.override.md`)	`CLAUDE.md`
Konfiguration	`~/.codex/config.toml`, Profile	`settings.json`, `.claude/`
Anpassbare Befehle	Custom Prompts (`~/.codex/prompts/`), Skills	Custom Commands, Skills, Subagents
Sandbox	macOS Seatbelt / Linux Landlock+seccomp / Windows AppContainer	eigenes Permission-/Sandbox-Modell
Cloud-Pendant	Codex Cloud (chatgpt.com/codex)	Claude-Code-Cloud-/Remote-Agents (je nach Plan)
Lizenz CLI	Open Source (Apache-2.0)	Anthropic-eigen

Nutzerberichte (mit Vorsicht zu genießen, da methodisch nicht immer transparent) attestieren Claude Code häufig sauberere, engere Diffs und idiomatischeren Code, während GPT-5.3-Codex bei Terminal-Bench-2.0-Benchmarks vorn liegt, insbesondere bei DevOps-, Skripting- und CLI-lastigen Aufgaben. Eine oft zitierte Faustregel aus Vergleichsartikeln lautet sinngemäß: „Claude Code für Code-Qualität, Codex CLI für Ausdauer bei Langläufen."

13. Umstieg von Claude Code: wie arbeitet man jetzt?

Ein typisches Claude-Code-Projekt legt seinen gesamten Anpassungsstand in einer `CLAUDE.md` und einem `.claude/`-Ordner ab:

```
mein-projekt/
├── CLAUDE.md                    ← Projektanweisungen (in Git)
├── CLAUDE.local.md               ← persönlich (NICHT in Git)
└── .claude/
    ├── settings.json              ← Permissions, Hooks, MCP-Server (in Git)
    ├── settings.local.json         ← lokal (NICHT in Git)
    ├── rules/                     ← modulare, teils per Glob gefilterte
Regeln
    ├── commands/                  ← eigene Slash-Commands
    ├── skills/                    ← komplexe, mehrstufige Workflows
    ├── agents/                    ← spezialisierte Subagenten
    └── hooks/                     ← Automation-Skripte, aus settings.json
aufgerufen
```

Codex kennt keinen direkten `.claude/`-Nachbau, sondern verteilt dieselben Konzepte auf `AGENTS.md`, `~/.codex/config.toml`, `~/.codex/prompts/` und das Skills-Feature. Die folgenden Abschnitte gehen jede Datei/jeden Ordner einzeln durch.

13.1 Überblick: `.claude/` → Codex-Äquivalente

Claude Code	Zweck	Codex-Entsprechung	Deckungsgrad
`CLAUDE.md`	Projektweite Anweisungen	`AGENTS.md`	fast 1:1
`CLAUDE.local.md`	persönliche, nicht geteilte Anweisungen	kein dokumentiertes 1:1-Äquivalent	teilweise, siehe 13.2
`.claude/rules/*.md` (+ Glob-Filter)	bereichsspezifische Regeln	verzeichnisweise verteilte `AGENTS.md`-Dateien	konzeptionell ähnlich, ohne Glob-Filter
`.claude/commands/*.md`	teambasierte Slash-Commands	Custom Prompts (`~/.codex/prompts/`) oder Skills	teilweise – Custom Prompts sind nicht repo-geteilt
`.claude/skills/*/SKILL.md`	komplexe, mehrstufige Workflows	Codex Skills	direkt vergleichbar
`.claude/agents/*.md` (Subagents)	spezialisierte, isolierte Rollen	Multi-Agent-Feature (`[features] multi_agent`)	konzeptionell ähnlich, deutlich weniger granular dokumentiert
`.claude/settings.json` → `permissions`	erlaubte/verbotene Aktionen	`approval_policy`, `sandbox_mode` in `config.toml`	anderes Modell, kein Regel-für-Regel-Mapping
`.claude/settings.json` → `mcpServers`	MCP-Server-Liste	`[mcp_servers.NAME]` in `config.toml`	direkt vergleichbar, anderes Dateiformat
`.claude/settings.json` → `hooks` +	erzwungene Automatisierung bei Events	Hooks-Feature (`[features] hooks`, `/hooks`)	vorhanden, aber Stand Juli 2026 wenig dokumentiert

Claude Code	Zweck	Codex-Entsprechung	Deckungsgrad
`.claude/hooks/*.sh`			

13.2 `CLAUDE.md` und `CLAUDE.local.md` → `AGENTS.md`

Der Inhalt einer `CLAUDE.md` lässt sich fast 1:1 in eine `AGENTS.md` im selben Verzeichnis übernehmen – beide Formate sind einfaches Markdown mit Projektkontext, Konventionen und Arbeitsanweisungen, beide werden vor jeder Aufgabe automatisch geladen. Wer beide Agenten parallel im selben Repository nutzt, kann entweder beide Dateien parallel pflegen oder eine als symbolischen Link auf die andere anlegen – inhaltlich sollten projektspezifische Vorgaben ohnehin in beiden Werkzeugen gleich sein.

Für `CLAUDE.local.md` (persönliche, nicht in Git eingecheckte Ergänzungen) gibt es kein offiziell dokumentiertes direktes Gegenstück. Am nächsten kommt `AGENTS.override.md`: Codex liest pro Verzeichnis zuerst `AGENTS.override.md`, dann `AGENTS.md`, wobei jeweils nur die erste nicht-leere Datei zählt (kein Zusammenführen beider). Praktisch heißt das: Eine lokale `AGENTS.override.md`, in `.gitignore` eingetragen, kann persönliche Ergänzungen aufnehmen – ersetzt dabei aber die projektweite `AGENTS.md` an diesem Ort komplett, statt sie zu ergänzen. Wer beide Inhalte kombinieren will, muss sie manuell in einer Datei zusammenführen.

13.3 `.claude/rules/` → verteilte `AGENTS.md`-Dateien

Claude Code erlaubt es, Regeln nach Themenbereich in `.claude/rules/*.md` aufzuteilen, teils mit Glob-Filtern, sodass z. B. nur beim Bearbeiten von Swift-Dateien die Swift-Style-Regel geladen wird. Ein direktes, glob-gefiltertes Pendant dokumentiert Codex nicht. Näherungsweise lässt sich derselbe Effekt über die Verzeichnis-Auflösung von `AGENTS.md` erreichen: Da Codex vom Projekt-Root abwärts bis zum aktuellen Arbeitsverzeichnis alle `AGENTS.md`-Dateien einsammelt und konkateniert, kann man bereichsspezifische Regeln einfach in die jeweiligen Unterverzeichnisse legen, z. B. `frontend/AGENTS.md` für Frontend-Konventionen oder `api/AGENTS.md` für API-Regeln. Der Unterschied: Diese Regeln greifen nur, wenn Codex tatsächlich in oder unterhalb dieses Verzeichnisses arbeitet – eine dateiendungsbasierte Filterung wie bei `.claude/rules/`-Globs gibt es nicht.

13.4 `.claude/commands/` → Custom Prompts bzw. Skills

Hier ist Vorsicht geboten, weil die naheliegende Übersetzung nicht ganz passt: Claude-Code-Custom-Commands unter `.claude/commands/` werden über Git mit dem gesamten Team geteilt. Codex' Custom Prompts unter `~/.codex/prompts/` sind dagegen rein lokal im Home-Verzeichnis abgelegt und werden nicht über das Repository geteilt – ein Teamkollege, der dasselbe Repo klont, bekommt sie nicht automatisch mit. Für teamweit geteilte, wiederverwendbare Befehle ist daher eher das Skills-Feature (13.5) die passendere Zielstruktur als Custom Prompts. Wer wirklich nur einen persönlichen, schnellen Slash-Befehl migrieren will (vergleichbar mit `~/.claude/commands/`, also den nutzerglobalen Claude-Code-Commands), kann dafür weiterhin `~/.codex/prompts/` verwenden.

13.5 `.claude/skills/` → Codex Skills

Codex Skills sind das direkteste Gegenstück zu Claude Codes `.claude/skills/*/SKILL.md`-Struktur: Beide sind teilbare, mehrstufige Workflow-Beschreibungen, die im Repository liegen und vom Agenten auch implizit aufgerufen werden können, wenn die Aufgabe inhaltlich dazu passt (nicht nur per expliziten Slash-Command wie bei Custom Commands/Prompts). Eine bestehende `SKILL.md` lässt sich inhaltlich meist direkt übernehmen; unterschiedlich ist vor allem der genaue Ablageort und ob Zusatzdateien (Beispiele, Checklisten) auf dieselbe Weise referenziert werden können – das sollte man bei komplexeren Skills im Zweifel gegen die aktuelle Codex-Doku prüfen.

13.6 `.claude/agents/` (Subagents) → Codex Multi-Agent-Feature

Claude Codes Subagents (`.claude/agents/*.md`) definieren spezialisierte, in einem eigenen Kontext arbeitende Rollen (z. B. ein reiner Code-Reviewer oder ein Datenschutz-Prüfer), die die Hauptsession gezielt aufrufen kann. Codex hat mit dem Feature-Flag `multi_agent` (`[features] multi_agent = true`) ein konzeptionell verwandtes Konstrukt; laut Changelog wurde in CLI-Version 0.145.0 (21. Juli 2026) ein „Multi-Agent-V2" mit konfigurierbaren Sub-Agent-Modellen stabilisiert. Die genaue Syntax, um einen Sub-Agenten wie in Claude Code per Markdown-Datei mit eigener Rollenbeschreibung, eigenem Werkzeugzugriff und eigenem Modell zu definieren, war zum Zeitpunkt der Recherche nicht in vergleichbarer Ausführlichkeit dokumentiert wie bei Claude Code. Wer stark auf Subagents mit klar abgegrenzten Rollen setzt, sollte vor dem Umstieg gezielt prüfen, ob Codex' Multi-Agent-Feature in der aktuell installierten Version bereits denselben Grad an Konfigurierbarkeit bietet – andernfalls bleibt zunächst nur eine einzelne `AGENTS.md` mit mehreren beschriebenen Rollen als Behelfslösung.

13.7 `.claude/settings.json`: Permissions, Hooks, MCP-Server

`.claude/settings.json` bündelt bei Claude Code drei Dinge, die bei Codex auf unterschiedliche Konfigurationsstellen verteilt sind:

- **Permissions** (welche Tools/Befehle ohne Rückfrage laufen dürfen) entsprechen bei Codex der Kombination aus `sandbox_mode` (`read-only` / `workspace-write` / `danger-full-access`) und `approval_policy` (`untrusted` / `on-request` / `never`) in `config.toml`. Das Modell ist gröber granular als Claude Codes Permission-Regeln (die einzelne Tools/Befehlsmuster erlauben oder verbieten können) – ein Regel-für-Regel-Mapping jeder einzelnen Permission-Zeile gibt es nicht, man muss die Gesamtstrenge neu abwägen.
- **`mcpServers`** lässt sich inhaltlich direkt übertragen: Serverbefehl, Argumente und Umgebungsvariablen aus der JSON-Struktur von `.claude/settings.json` bzw. `.mcp.json` müssen aber manuell in eine `[mcp_servers.NAME]`-Tabelle in `config.toml` umgeschrieben werden (JSON → TOML, kein automatischer Import). Für HTTP-/Streamable-MCP-Server gibt es ein eigenes Tabellenformat mit `url` und `bearer_token_env_var` statt `command`/`args`.
- **Hooks** (`.claude/settings.json` → `hooks`, plus die referenzierten Skripte in `.claude/hooks/*.sh`) sind konzeptionell auch in Codex vorhanden (Feature-Flag `hooks`, `/hooks`-Command in der TUI), aber die genaue Event-/Konfigurationssyntax war zum Zeitpunkt der Recherche nicht vollständig offiziell dokumentiert (siehe auch 8.5). Vorhandene Claude-Code-Hook-Skripte (z. B. ein Skript, das Health-Daten-Dateien vor Bearbeitung schützt) lassen sich als reine Shell-Skripte grundsätzlich weiterverwenden – die Einbindung in Codex muss aber pro CLI-Version neu geprüft werden, statt blind übernommen zu werden.

13.8 Schritt-für-Schritt-Checkliste

1. `CLAUDE.md` inhaltlich nach `AGENTS.md` kopieren (13.2); bei Bedarf `CLAUDE.local.md`-Inhalte manuell einarbeiten oder in eine `.gitignore`-geschützte `AGENTS.override.md` auslagern.
2. Bereichsspezifische `.claude/rules/*.md` in die passenden Unterverzeichnisse als eigene `AGENTS.md`-Dateien verteilen (13.3).
3. Team-Custom-Commands aus `.claude/commands/` als Codex Skills nachbauen, rein persönliche Commands wahlweise als `~/.codex/prompts/`-Dateien (13.4).
4. `.claude/skills/*/SKILL.md` möglichst direkt in Codex Skills übernehmen (13.5).
5. `.claude/agents/*.md` (Subagents) gegen den aktuellen Funktionsumfang des Codex-Multi-Agent-Features prüfen, statt blind 1:1 zu migrieren (13.6).
6. `.claude/settings.json` → `mcpServers` manuell nach `config.toml` übertragen, Permission-Strenge neu festlegen (`sandbox_mode`/`approval_policy`), Hooks einzeln testen (13.7).

7. Am Ende beide Konfigurationen (Claude Code und Codex) im selben Repository gegen dieselben Testfälle laufen lassen, um Abweichungen im Verhalten frühzeitig zu bemerken – insbesondere bei Permission-/Sandbox-Strenge, da hier keine 1:1-Übersetzung existiert.

13.9 Direkter Umstiegsweg: `/import`

Seit CLI-Version 0.145.0 (21. Juli 2026) gibt es experimentell den Befehl `/import`, der Einstellungen aus Cursor und Claude Code automatisiert übernehmen soll. Da dieses Feature noch als experimentell markiert ist und öffentlich nicht im Detail dokumentiert war, welche der oben genannten Dateien/Ordner es tatsächlich abdeckt (vermutlich zunächst vor allem `CLAUDE.md` → `AGENTS.md`, nicht notwendigerweise Subagents oder Hooks), empfiehlt sich vor dem produktiven Einsatz in jedem Fall eine manuelle Kontrolle der übernommenen Einstellungen anhand der Checkliste aus 13.8.

14. Datenschutz und Sicherheit

Sandbox: Standardmäßig hat Codex keinen ausgehenden Netzwerkzugriff und darf nur innerhalb des freigegebenen Workspace schreiben (siehe 7.2). Auf macOS wird dies über Seatbelt-Profile durchgesetzt, unter Linux über Landlock (Dateisystem) plus seccomp-BPF (blockiert u. a. `connect`/`bind`/`listen`/`sendto`; lokale AF_UNIX-Sockets sind ausgenommen). Ein dokumentierter Bug (GitHub-Issue #10390) zeigte, dass `network_access = true` in der Konfiguration unter macOS zeitweise stillschweigend ignoriert wurde – ein Beispiel dafür, dass Sandbox-Konfiguration nicht blind vertraut, sondern stichprobenartig geprüft werden sollte.

Datenverarbeitung/Training: Für Business-, Enterprise- und API-Kunden gilt standardmäßig, dass Ein- und Ausgaben nicht zum Training verwendet werden, sofern kein ausdrückliches Opt-in erfolgt. Codex hat zusätzlich einen eigenen Schalter für „Training auf vollständige Umgebungen", unabhängig vom allgemeinen ChatGPT-Data-Controls-Schalter. Ein genereller Opt-out ist über das OpenAI Privacy Portal bzw. „Data Controls" in den ChatGPT-Einstellungen möglich. Auch bei aktiviertem Opt-out bleibt eine Datenspeicherung zu Missbrauchsprävention und Betriebszwecken bestehen (bei API/Enterprise standardmäßig bis zu 30 Tage), sofern nicht **Zero Data Retention (ZDR)** konfiguriert ist – ZDR ist für berechtigte API-Endpunkte verfügbar, insbesondere für regulierte Branchen wie Gesundheitswesen und Finanzen.

Bekannt gewordene Sicherheitsvorfälle/-bedenken (chronologisch, mit Vorsicht bei Detailangaben zu lesen):

- **CVE-2025-61260** (Dezember 2025, CVSS 9,8, kritisch): Command-Injection über automatisch geladene MCP-Server-Konfigurationen ohne interaktive Freigabe – laut Berichten vor Public Disclosure gepatcht.
- **PromptArmor** (April 2026): indirekte Prompt-Injection in der Codex-Desktop-App, die ohne Nutzerinteraktion sensible E-Mail-Inhalte exfiltrieren konnte.
- Berichte über **GitHub-Token-Diebstahl** via Command-Injection in Cloud-Sandbox-Tasks (kurzlebige OAuth-Tokens betroffen).
- **Cymulate-Bericht**: Remote Code Execution über die `web.run`-Websuchfunktion kombiniert mit Prompt Injection und Binary-Hijacking, primär auf Windows.
- **Backslash-Security-Bericht** zu „AGENTS.md Injection": versteckte Anweisungen in einer projektinternen `AGENTS.md` können zu stillem Credential-Diebstahl führen – ein Angriffsvektor, der besonders relevant ist, wenn man fremde Repositories mit bereits vorhandener `AGENTS.md` öffnet.

Praktische Konsequenz für Nutzer: Wie bei jedem agentischen Coding-Tool sollte man fremde Repositories nicht unbesehen als „vertrauenswürdig" einstufen, MCP-Server-Einträge und `AGENTS.md`-Inhalte vor der ersten Ausführung prüfen und produktiv nach Möglichkeit mit eingeschränkter Sandbox statt `danger-full-access` arbeiten.

15. Vorteile und Nachteile

Stärken:

- Terminal-natives, chat-getriebenes Arbeiten direkt im Repository, kombiniert mit voller Datei- und Kommandoausführung.
- Sehr niedrige Einstiegshürde: Login per ChatGPT-Konto oder API-Key genügt, kein separates Abo nötig, wenn ohnehin ein bezahlter ChatGPT-Plan besteht.
- Starke GitHub-Integration (automatische PR-Erstellung, PR-Reviews, Reaktion auf Issue-Erwähnungen).
- Codex Cloud erlaubt echte Parallelisierung mehrerer Aufgaben, ohne die lokale Maschine zu belasten.
- Laut Terminal-Bench-2.0-Zahlen besonders stark bei DevOps-, Skripting- und CLI-lastigen Aufgaben.
- CLI ist quelloffen (Apache-2.0) und lässt sich auch mit reinem API-Key ohne ChatGPT-Abo nutzen.

Schwächen/Kritikpunkte:

- Unklare, variable Wartezeiten bei Cloud-Tasks.
- Für Cloud-Sandboxes muss vorab eine Environment-Konfiguration definiert werden – unpraktisch, wenn Testumgebungen spontan hochgefahren werden müssten.
- Umstellung auf Token-/Credit-basierte Abrechnung (April 2026) macht Kosten schwerer vorhersehbar als bei einer reinen Nachrichten-Flatrate.
- Setup-Komplexität; Vorschläge sind nicht immer optimal auf projektspezifische Ziele abgestimmt.
- Mehrere kritische Sicherheitslücken 2025/2026 (siehe Kapitel 14) haben in Teilen der Security-Community Vertrauen gekostet, insbesondere bezüglich automatischer MCP-Ausführung und AGENTS.md-Injection.
- Windows-Support gilt weiterhin als experimentell; produktiv wird WSL2 empfohlen.

16. Bekannte Limitierungen

Kontextfenster: Für die Subscription-/CLI-Nutzung gilt laut einer detaillierten Analyse ein Gesamtbudget von rund 400.000 Token (davon 272.000 für Eingabe, 128.000 für Ausgabe reserviert; die CLI hält zusätzlich rund 5 % Sicherheitsabstand, sodass effektiv etwa 258.000 Eingabe-Token nutzbar bleiben). Das API-seitige Kontextfenster von GPT-5.5 reicht laut Anbieterangaben zwar bis zu 1,05 Mio. Token, dieser höhere Wert gilt aber nicht automatisch für das CLI-Subscription-Budget. In den GitHub-Issues des Projekts finden sich wiederholt Nutzerbeschwerden über spürbare Schwankungen des effektiv verfügbaren Kontextfensters. Bei großen Monorepos oder API-lastigem Code empfiehlt es sich, mit `/compact` frühzeitig zusammenzufassen bzw. die Auto-Compaction-Schwelle abzusenken.

Große Repositories: Repo-weite Analysen, lange Refactorings und tool-lastige Multi-Agent-Sessions hängen stark von einer stabilen Kontextkapazität ab – ein Bereich, in dem laut Nutzerrückmeldungen noch Verbesserungsbedarf besteht.

Windows: Offiziell „unterstützt" sind in erster Linie macOS und Linux. Die native Windows-Installation mit AppContainer-Sandbox funktioniert seit Anfang 2026, gilt aber weiterhin als experimentell; produktiv wird WSL2 empfohlen, weil es dieselbe Landlock/seccomp-Sandbox wie natives Linux bereitstellt.

17. FAQ

17.1 Brauche ich einen ChatGPT-Plan, oder reicht ein API-Key?

Beides funktioniert unabhängig voneinander. Mit `codex login` per ChatGPT-Konto wird das im jeweiligen Plan (Plus/Pro/Business/Enterprise/Edu) enthaltene Kontingent verwendet; mit einem API-Key von platform.openai.com zahlt man stattdessen nach tatsächlichem Token-Verbrauch zu API-Preisen, ganz ohne ChatGPT-Abo.

17.2 Was passiert mit meinem Code – wird er zum Training genutzt?

Für Business-, Enterprise- und API-Zugänge gilt standardmäßig kein Training auf Ein-/Ausgaben, sofern kein Opt-in erfolgt. Für private ChatGPT-Konten lässt sich das Training über „Data Controls" in den Einstellungen deaktivieren. Details siehe Kapitel 14.

17.3 Läuft Codex CLI auch offline?

Nein – die Modellinferenz läuft ausschließlich über OpenAIs Server, ein lokales Modell (`--oss`-Flag für Open-Source-Modelle) ist die einzige Ausnahme, in der zumindest ein Teil der Verarbeitung lokal laufen kann. Für die Standard-Codex-Modelle ist eine Internetverbindung zwingend erforderlich.

17.4 Was ist der Unterschied zwischen Custom Prompts und Skills?

Custom Prompts sind einfache, rein lokale Markdown-Dateien unter `~/.codex/prompts/`, die nur explizit per Slash-Command aufgerufen werden. Skills sind der Nachfolger: teilbar über das Repository und von Codex auch implizit aufrufbar, wenn die Aufgabe inhaltlich dazu passt.

17.5 Kann ich meine Cursor- oder Claude-Code-Einstellungen automatisch übernehmen?

Seit CLI-Version 0.145.0 (Juli 2026) existiert experimentell der Befehl `/import` für genau diesen Zweck (siehe 13.4). Da das Feature neu und experimentell ist, empfiehlt sich eine manuelle Kontrolle nach der Übernahme.

18. Vergleich: Codex, Claude Code, Cursor & Co.

18.1 Vergleichstabelle

	Codex CLI	Claude Code	Cursor
Anbieter	OpenAI	Anthropic	Anysphere
Typ	Terminal-Agent	Terminal-Agent	eigenständige IDE (VS-Code-Fork)
Lizenz/Quelloffenheit	Open Source (Apache-2.0)	proprietär	proprietär
Kontextdatei	`AGENTS.md`	`CLAUDE.md`	`.cursor/rules`
Modellauswahl	nur OpenAI-Modelle	nur Anthropic-Modelle	mehrere Anbieter parallel (Claude, GPT, Gemini u. a.)
Cloud-Pendant	Codex Cloud	Cloud-/Remote-Agents (planabhängig)	Cloud Agent
Preis Einstieg	ab 20 $/Monat (Plus) bzw. reiner API-Verbrauch	Claude-Abo bzw. API-Verbrauch	ab 20 $/Monat (Pro)

18.2 Codex CLI

Stärke: offen, eng mit dem ChatGPT-Ökosystem und GitHub verzahnt, laut Benchmarks besonders stark bei DevOps-/CLI-lastigen Aufgaben und bei lang laufenden, parallelisierbaren Cloud-Tasks.

18.3 Claude Code

Stärke: laut mehreren Vergleichsberichten tendenziell sauberere, engere Diffs und idiomatischerer Code; enges Zusammenspiel mit dem restlichen Claude-Ökosystem (Skills, Subagents, Hooks). Details siehe `Tutorial-claude-code-guide/claude-code-guide-DE.md`.

18.4 Cursor

Anders als Codex CLI und Claude Code keine reine Terminal-Anwendung, sondern eine vollständige IDE (VS-Code-Fork) mit Tab-Completion, Composer und Multi-Modell-Zugriff auf verschiedene Anbieter inklusive Codex- und Claude-Modelle. Details siehe `Tutorial-Cursor/cursor-de.md`.

18.5 Weitere nennenswerte Agenten

- **Gemini CLI** (Google) – ebenfalls ein terminalbasierter Coding-Agent mit Dateizugriff und Multi-Step-Tasks.
- **GitHub Copilot CLI/Coding Agent** – seit Mai 2026 auf tokenbasierte Abrechnung statt Flatrate-„Premium Requests" umgestellt, was Kritik wegen unvorhersehbarer Kosten bei langen Sessions auslöste.
- **Amazon Q Developer** – Stärke bei AWS-Infrastruktur/IaC (IAM, CDK).

Zu Umbenennungen dieser Konkurrenzprodukte kursieren im Netz teils widersprüchliche, sehr aktuelle Angaben aus einzelnen Sekundärquellen – vor einer Übernahme in eigene Entscheidungen empfiehlt sich ein Blick auf die jeweilige offizielle Produktseite.

18.6 Fazit

Codex CLI und Claude Code sind sich als reine Terminal-Agenten strukturell am ähnlichsten und daher am direktesten vergleichbar; Cursor spielt in einer anderen Kategorie (vollständige IDE mit Modellauswahl).

Wer bereits tief im ChatGPT-/GitHub-Ökosystem arbeitet oder Wert auf offenen CLI-Quellcode legt, findet in Codex CLI eine naheliegende Wahl; wer bereits Claude-Abo-Nutzer ist oder besonders auf saubere, kompakte Diffs Wert legt, bleibt tendenziell eher bei Claude Code. Ein produktiver Parallelbetrieb beider Werkzeuge im selben Projekt ist unproblematisch, solange `AGENTS.md` und `CLAUDE.md` inhaltlich synchron gehalten werden.

19. Quellen

Diese Anleitung basiert auf einer Web-Recherche (Stand Juli 2026). Die wichtigsten verwendeten Quellen, gegliedert nach Themenblock:

Codex – offizielle Dokumentation (developers.openai.com/codex, leitet Stand Juli 2026 per Redirect auf learn.chatgpt.com/docs weiter):

- openai/codex – GitHub-Repository
- Quickstart
- Custom instructions with AGENTS.md
- Config basics
- Model Context Protocol
- Custom Prompts
- Slash commands in Codex CLI
- Developer Commands Reference
- Codex cloud
- Codex IDE extension
- Codex app
- Sandboxing (Concepts)
- Authentication
- Permissions
- Agent approvals & security
- Pricing
- Models
- Codex changelog

OpenAI – Blog/Ankündigungen:

- Introducing Codex
- Codex is now generally available
- Introducing the Codex app
- Codex for (almost) everything
- Introducing GPT-5.2-Codex
- Previewing GPT-5.6 Sol
- GPT-5.6
- Enterprise privacy at OpenAI
- Business data privacy, security, and compliance
- Data controls in the OpenAI platform

OpenAI Help Center:

- How your data is used to improve model performance
- Data Controls FAQ
- Codex rate card
- Moving to the new ChatGPT desktop app
- ChatGPT Work and Codex
- What are the system requirements for the ChatGPT macOS app

Desktop-App – Presse/Community (Launch, Review, Kritik):

- TechCrunch: OpenAI launches new macOS app for agentic coding
- VentureBeat: OpenAI launches a Codex desktop app for macOS
- reconn-ai.com: ChatGPT July 9, 2026 – the new ChatGPT desktop app brings Chat, Work and Codex together

- thecartine.substack.com: Codex App Doesn't Suck (Review)
- BSWEN Blog: Codex App Missing Features
- GitHub Issue #2000: Windows-Auth-Probleme

Unternehmen/Governance (OpenAI):

- OpenAI Codex (AI agent) – Wikipedia
- Who Owns OpenAI? – aifundingtracker.com
- OpenAI completes restructure, solidifying Microsoft as a major shareholder – CNBC
- OpenAI's For-Profit Pivot in 2026 – Skycrumbs

Geschichte / Rust-Rewrite / Presse:

- Taskade: History of OpenAI Codex
- Kunalganglani: What happened to OpenAI Codex
- Visual Studio Magazine: The Return of Codex AI as an Agent
- OpenAI debuts Codex CLI – TechCrunch
- OpenAI launches Codex – AlternativeTo
- The codex-rs Architecture: How OpenAI Rewrote Codex CLI in Rust
- OpenAI Fully Transitions to Rust for Refactoring Codex CLI
- JetBrains Blog: Codex Is Now Integrated Into JetBrains IDEs
- Neowin: OpenAI's Codex extension is now available for JetBrains IDEs

Installation/Konfiguration – Community/Drittanbieter:

- Inventive HQ: Avoid the npm Trap
- Codex Danielvaughan Knowledge Base: Official Documentation Guide
- Codex Knowledge Base: MCP Server Management

Sandbox/Sicherheit:

- Sandboxing Implementation – DeepWiki
- Research: OpenAI Codex CLI Sandbox Implementation Analysis – Simon Willison
- macOS network access silently ignored – GitHub Issue #10390
- Inside the Codex Sandbox: Platform-Specific Implementation
- Codex CLI on Windows: Native Sandbox, WSL Integration
- OpenAI Codex CLI Command Injection Vulnerability – Cyberpress
- OpenAI Codex CLI Vulnerability: Command Injection – Check Point Research
- OpenAI Codex vulnerability enabled GitHub token theft – SiliconANGLE
- Remote Code Execution in Codex CLI via Prompt Injection – Cymulate
- AGENTS.md Injection in OpenAI Codex CLI – Backslash Security

Preise – Drittanbieter/Aggregatoren (gegen offizielle Pricing-Seite gegenzuchecken):

- OpenAI Codex pricing in 2026 – eesel AI
- ChatGPT Codex Pricing (July 2026) – Automation Atlas
- Codex Pricing and Usage Limits (July 2026) – morphllm.com
- GPT-5.3-Codex – API Pricing & Benchmarks – OpenRouter
- GPT 5 Codex API Pricing 2026 – pricepertoken.com

Vergleich mit anderen Agenten:

- Cursor vs Claude Code vs Codex (2026) – morphllm.com
- Claude Code vs Codex vs Cursor – cosmicjs.com
- OpenAI Codex vs Cursor vs Claude Code – nxcode.io
- AI Coding Agents Compared 2026 – ofox.ai
- GitHub Copilot vs Amazon Q Developer vs Gemini Code Assist – arnav.au
- Top 5 CLI coding agents in 2026 – Pinggy Blog

Nutzererfahrungen/Reviews:

- OpenAI Codex hands-on review – Hacker News
- OpenAI Codex CLI – cobusgreyling.substack.com
- OpenAI Codex Review – darkomedin.substack.com

Kontextfenster/Limitierungen:

- Increase Codex CLI context window – GitHub Issue #9857
- Support 1M token context for GPT-5.5 in Codex – GitHub Issue #19464
- SEVERE REGRESSION context cut – GitHub Issue #32806
- GPT-5.5's Million-Token Context Window – codex.danielvaughan.com
- How to Run OpenAI Codex in Windows with WSL – apidog.com

Hinweis zur Recherchequalität

Codex verändert sich derzeit außergewöhnlich schnell – allein zwischen Dezember 2025 und Juli 2026 wurden mehrere Modellgenerationen (GPT-5-Codex bis GPT-5.6 Sol/Terra/Luna) und mehrere CLI-Versionssprünge veröffentlicht. Folgende Punkte in dieser Anleitung sind nur über Drittanbieter-Blogs, Preis-Aggregatoren oder einzelne Sekundärquellen belegt, nicht über eine offizielle Erstquelle, und sollten vor einer produktiven Entscheidung noch einmal direkt geprüft werden (z. B. per `codex --help`, `codex doctor`, oder auf der jeweils aktuellen offiziellen Doku-Seite):

- die genaue Modellbezeichnung, Verfügbarkeit und Preisstruktur der GPT-5.6-Familie (Sol/Terra/Luna) sowie sämtliche Dollar-/Credit-Beträge pro Million Token,
- die exakten Nachrichten-Kontingente pro 5-Stunden-Fenster je Plan und Modell,
- die genaue Hooks-Syntax und -Semantik in der aktuellen CLI-Version,
- die Existenz und Funktionsweise einer `.codexignore`-Datei (in den durchsuchten Quellen nicht bestätigt),
- der genaue Funktionsumfang des experimentellen `/import`-Befehls (Stand CLI 0.145.0),
- Details zur Desktop-App-Verschmelzung mit der ChatGPT-Desktop-App zum 9. Juli 2026 (Chat/Work/Codex-Struktur, „ChatGPT Classic"): Die zentralen `help.openai.com`-Artikel dazu waren beim Abruf per Web-Fetch blockiert (HTTP 403), die hier verwendeten Angaben stammen aus Zusammenfassungen Dritter, die diese Artikel zitieren,
- die vollständige Liste der Tastenkürzel und Einstellungen der Desktop-App sowie einzelne, nur in Community-Blogposts belegte Kritikpunkte (z. B. fehlendes `--add-dir`-Äquivalent, instabile Git-Diffs),
- die Umbenennungen konkurrierender Produkte (z. B. „Gemini CLI → Antigravity", „Amazon Q Developer → Kiro CLI") – diese stammen aus einzelnen, ungewöhnlich spezifischen Sekundärquellen und sollten vor Übernahme gegen offizielle Google-/AWS-Ankündigungen geprüft werden,
- der genaue Konfigurationsumfang von Codex' Multi-Agent-Feature (`[features] multi_agent`) im Vergleich zu Claude Codes Subagents (Kapitel 13.6) – hierzu lag zum Zeitpunkt der Recherche keine mit Claude Code vergleichbar ausführliche offizielle Dokumentation vor,
- ob und in welchem Umfang `.claude/rules/`-Glob-Filter, `.claude/settings.json`-Permission-Regeln und Hook-Skripte durch den experimentellen `/import`-Befehl automatisch mit übernommen werden (Kapitel 13.9) – dazu wurde keine detaillierte offizielle Feature-Liste gefunden.

Die grundsätzliche Architektur (CLI/Cloud/IDE-Extensions/Desktop-App, AGENTS.md, config.toml, MCP-Unterstützung, Sandbox-Mechanismus) sowie die 2025/2026 bekannt gewordenen Sicherheitslücken gelten dagegen als über mehrere unabhängige Quellen (offizielle Doku, GitHub-Issues, Sicherheitsforschungsberichte) übereinstimmend belegt.