ENTSCHEIDUNGSHILFE · FÜR BENJAMIN

Reverse Engineering mit Codex

Neun Repositories aus Arnies Video eingeordnet
Entscheidungshilfe für deinen Einstieg unter Windows · 7. Oktober 2026

Meine Empfehlung: ein kleiner, überprüfbarer Start

Starte mit ILSpy an einer eigenen .NET-Demo. Das gibt dir unter Windows schnell ein verständliches Ergebnis: C#-Code aus einer bekannten DLL, den du mit dem Original vergleichen kannst. Codex kann die Ausgabe erklären und Tests entwerfen. Für diesen Weg ist kein zusätzlicher MCP-Server zwingend nötig.

Für einen echten Codex-Analyseworkflow ist REA der interessanteste nächste Versuch. Es verbindet Analysewerkzeuge mit dem Agenten und dokumentiert Fähigkeiten und Testgrenzen relativ genau. Der native Windows-Pfad ist aber experimentell und eng begrenzt. Er ist kein universeller Spiele-Rückübersetzer.

- Zuerst lesen: ai-game-modding-guides für Planung, kleine Schritte, Logs und überprüfbare Tests.
- Danach praktisch: eigene .NET-DLL mit ILSpy; eine Funktion erklären und gegen bekannte Testfälle prüfen.
- Bei nativen x64-EXEs: Ghidra zunächst manuell kennenlernen; anschließend REA oder Ghidra-MCP kontrolliert ergänzen.
- Bei konkretem Spiel-Modding: Universal Modder als Werkzeugkasten, möglichst über eine vorhandene Mod-API.
- IDA-MCP lohnt sich vor allem mit passender vorhandener IDA-Lizenz; Cpp2IL erst bei einem konkreten Unity-IL2CPP-Ziel.
- AnyPS5 und PortPS5 sind anspruchsvolle Kompatibilitätsforschung, keine empfehlenswerten Anfängerprojekte.

Was diese Empfehlung trägt

Alle neun in der Videobeschreibung verlinkten Repositories wurden lesend geprüft. Sie erfüllen unterschiedliche Aufgaben: Dokumentation, Agent-Orchestrierung, MCP-Brücken, Decompilation und Konsolen-Kompatibilität. Deshalb wäre eine einzige Rangliste irreführend.

Keine Praxistest-Garantie: Es wurden weder Repository-Programme installiert oder ausgeführt noch echte Spiele analysiert. Aussagen zur Umsetzung stammen aus Dokumentation und ausgewählten Quelldateien. Der Bericht unterscheidet diese Befunde von Projektbehauptungen und eigenen Empfehlungen.

Die wichtigste Videokorrektur: Arnies WoW-Demo verändert Figuren-Assets. Sie zeigt keinen vollständigen Nachbau von World of Warcraft.

Was das Video tatsächlich zeigt

Ein gelungenes Modding-Beispiel, kein vollständiger Spielnachbau

Arnie zeigt Figuren aus World of Warcraft mit neuen Modellen. Bei 10:18 erklärt er ausdrücklich, dass er nicht die gesamte Windows-EXE dekompiliert hat. Analysiert und verändert wurden Figuren-Binärdaten; dazu kamen ein Blender-Modell und ein Patch. Ab 11:21 bleiben Welt, Physik, Skelett und Animationen original. Das ist Asset-Modding: vorhandene Spieldaten gezielt verstehen und ersetzen. [V1]

Die Demo veranschaulicht, dass ein KI-Agent viel Such-, Skript- und Prüfarbeit übernehmen kann. Sie beweist nicht, dass Codex beliebige proprietäre Anwendungen vollständig rekonstruieren kann. Ein Client enthält außerdem nicht automatisch den Code seiner serverseitigen Dienste.

Die Begriffe sauber trennen

- Disassembler: übersetzt Maschinenbefehle in lesbare Assemblersprache.
- Decompiler: erzeugt eine höherstufige Darstellung. Originalnamen, Kommentare und Projektstruktur müssen daraus nicht zurückkehren.
- MCP-Server: verbindet ein vorhandenes Analysewerkzeug mit einem Agenten. Er ist nicht selbst automatisch ein Decompiler.
- Reimplementierung: eine neue Implementierung mit überprüfbarem Verhalten. Das ist mehr Arbeit als das Anzeigen dekompilierten Codes.
- Asset-Modding: verändert beispielsweise Modelle, Texturen oder Datenformate, während große Teile des Programms unverändert bleiben.

Auch der Benchmark ist enger als die große These

Die aktuelle OpenAI-Primärquelle nennt für GPT-6 Astra auf SRE-Bench 88,0 % bei einem Versuch und 99,2 % innerhalb von vier Versuchen. Arnies ungefähre 60-%-Angabe stimmt damit nicht überein. Der Benchmark betrifft Programmverständnis und Rekonstruktion von Kernlogik; diese Werte sind keine Erfolgsquote für den vollständigen Nachbau beliebiger Spiele. Ältere Paper-Ergebnisse beziehen sich auf andere Modelle und Auswertungen. [V2, V3]

Der praktisch wertvolle Kern

Gib dem Agenten ein begrenztes Ziel und einen objektiven Test. Lass ihn analysieren, Hypothesen dokumentieren, Änderungen prüfen und Fehler korrigieren. Ein gutes Abbruchkriterium ist wichtiger als ein Agent, der unbegrenzt weiterarbeitet.

Welches Werkzeug für welches Ziel?

Die Einordnung bezieht sich auf deinen Einstieg mit Codex unter Windows. „Dokumentiert“ bedeutet nicht, dass die Verbindung hier praktisch getestet wurde.

Repository / Rolle	Codex-Weg und Windows	Geeignet als ...
Universal Modder Modding-Werkzeugkasten	Codex-Plugin vorhanden; Python, Windows-/WSL-Hilfen. Bash-Launcher und Zusatzdienste beachten.	Ergänzung für ein konkretes Modding-Ziel
REA Analyse-Orchestrierung	Codex/MCP dokumentiert. Nativer Windows-Ghidra-Pfad experimentell und eng begrenzt.	Späterer Codex-Analysebaustein
AI Game Modding Guides Lernmaterial	Markdown und Vorlagen; kein eigener Adapter oder Decompiler.	Erste Lektüre und Checkliste
Ghidra-MCP MCP-Brücke	Windows dokumentiert; Ghidra, Java und Python erforderlich. Codex-Praxis ungeprüft.	Native Analyse nach Ghidra-Grundlagen
IDA-MCP MCP-Brücke	Offizielle Codex-Anbindung; IDA 9.4+, Python und passende Lizenz.	Erweiterung für IDA-Nutzer
ILSpy .NET-Decompiler	GUI/CLI; Codex liest erzeugte Dateien. Kein belegter MCP im Kernrepo.	Bester einfacher eigener .NET-Versuch
Cpp2IL Unity-IL2CPP-Recovery	CLI, Windows-x86/x64-Pfad dokumentiert; unfertige Neufassung.	Spezialwerkzeug für eigenes IL2CPP-Ziel
AnyPS5 PS5-Kompatibilität	C++20-/GPU-Toolchain; Codex unterstützt Codeverständnis.	Fortgeschrittene Forschungslektüre
PortPS5 Windows-Hardfork	C++23, MinGW, Vulkan; offene Integration, keine Releases.	Architektur- und Testfallstudie

Zuerst den Dateityp bestimmen

Eine EXE oder DLL sagt noch nicht, welcher Decompiler passt. Normale .NET-Assemblies enthalten IL; native C/C++-Programme enthalten Maschinencode; Unity-IL2CPP benötigt einen eigenen Analyseweg. Ein fairer Test vergleicht Werkzeuge nur bei passenden Zieltypen.

Agenten und Modding-Workflows

1 · Universal Modder

Was vorhanden ist: Agent-Skills, Wissenssammlung und ausführbare Python-Hilfen, etwa für Scan, Assets, Sprites, Backups, Rendering und Windows-Steuerung. Der Scanner erkennt Engines und managed/native PE anhand von Dateien und Binärmerkmalen. Diese Heuristiken sind nützlich, belegen aber keinen funktionierenden Modding-Weg für jede Spielversion. Es ist kein neuer universeller Decompiler. [R1, A1]

Codex und Aufwand: Ein eigenes Codex-Plugin ist vorhanden. Die geprüfte MCP-Konfiguration bindet fal für Asset-Erzeugung an, keinen universellen RE-Server. Python ab 3.10 sowie Pillow, NumPy und PyYAML bilden die Basis; je nach Aufgabe kommen Blender, ffmpeg, Loader und SDKs hinzu. Der Bash-Launcher ist unter reinem PowerShell kein reibungsloser Ein-Klick-Weg. MIT. [A2]

Qualität und Risiko: Offline-Tests existieren; Windows- und Netzwerkaufrufe sind teilweise gemockt, die CI läuft auf Ubuntu. Die Windows-Komponente kann Prozesse und Eingaben steuern sowie Registry-Werte ändern. Agent-Sicherheitsregeln ersetzen keine Sandbox. fal kann Kosten und Datenübermittlung verursachen. [A3]

Urteil: Für ein bestimmtes, offiziell moddbares Spiel brauchbar. Zunächst Scan und Plan auf einer Kopie; Installationen, Cloud-Assets und Automatisierung gesondert prüfen.

2 · REA

Was vorhanden ist: CLI-/MCP-Orchestrierung mit echten Analyseanbindungen. Die Ghidra-Brücke liest Funktionen, Pseudocode, Referenzen und Symbole, prüft die Ziel-Hashidentität und authentifiziert lokale Kommunikation. Codex ist dokumentiert. Ein separater JavaScript-/Electron-Pfad benötigt keine Ghidra-Installation. MIT. [R2, A4]

Die entscheidende Windows-Grenze: Experimentelles Windows 10+ x64, feste lokale NTFS-Volumes, Ghidra 12.1.x und passendes 64-Bit-JDK. Der native Ghidra-Pfad akzeptiert x86-64-PE-Anwendungen, ausdrücklich keine DLLs oder managed Assemblies. Seine 25 read-only Operationen sind keine allgemeine Windows-Parität. .NET hat einen separaten Pfad. [A5]

Stand und Risiko: Dokumentierte npm-Veröffentlichung 5.0.0; main kann neuer sein. Umfangreiche Unit-, Boundary-, MCP- und Provider-Tests sind dokumentiert, aber hier nicht ausgeführt. Schutzmaßnahmen wie Loopback-Token und Windows-DACLs sind keine Ghidra-Sandbox. Auch lokale Analyseausgaben können durch einen verbundenen Cloud-Agenten weiterverarbeitet werden. [A6]

Urteil: Der vielversprechendste Kandidat für deinen späteren Codex-Analyseworkflow. Mit einer kleinen eigenen Datei und einer konkreten Frage beginnen; native Windows-Einrichtung bleibt anspruchsvoll.

Mit einem guten Arbeitsablauf anfangen

3 · AI Game Modding Guides

Was vorhanden ist: Dokumentation, Fallstudien und Projektvorlagen. Der geprüfte Dateibaum enthält Markdown, Templates, Lizenz und .gitignore, aber keine ausführbare Analyseengine oder automatisierte Testsuite. Das Projekt bezeichnet sich als Entwurf. Codex wird als möglicher Agent genannt; einen eigenen Codex-Adapter gibt es nicht. [R3]

Der größte Nutzen: Erst recherchieren und planen, dann kleine Änderungen mit Git-Zwischenständen, Logs und Übergabedokumenten. Vorlagen für Schnittstellenverträge und Spieltests helfen, aus einer Idee überprüfbare Kriterien zu machen. Für einen Anfänger ist diese Arbeitsweise wichtiger als eine lange Liste angeschlossener Tools. [A7]

Grenzen: Community-Erfahrungen sind keine kontrollierte Vergleichsstudie. Modell- und Produktangaben können schnell altern. Ein manueller Testleitfaden bestätigt weder die Funktionsfähigkeit verlinkter Projekte noch ihre Sicherheit. MIT gilt für den Repository-Text, nicht automatisch für Spielassets oder andere verlinkte Inhalte. [A8]

Urteil: Die beste erste Lektüre in dieser Auswahl. Benutze sie als Checkliste und Inspiration, nicht als Zusage, dass umfassende Spiel-Rewrites einfach oder zuverlässig werden.

Eine sinnvolle Schleife mit Codex

- Frage eingrenzen: Was soll genau herausgefunden werden?
- Belege sammeln: Dateityp, Hash, Strings, Referenzen und relevante Funktion.
- Hypothese formulieren: Was macht die Funktion vermutlich, was ist noch unklar?
- Vorhersage testen: Mit eigenen festgelegten Eingaben gegen das Original vergleichen.
- Abweichungen festhalten: Unsicherheit nicht durch eine überzeugend klingende Erklärung verdecken.
- Erst dann erweitern: Einen weiteren Fall hinzufügen oder genau eine kleine Änderung machen.

Warum nicht sofort „baue mir das ganze Spiel nach“?

Diese Aufgabe vermischt Dateiformate, Programmverhalten, Assets, Laufzeitdienste und oft externe Server. Ein Agent kann lange produzieren, ohne dass das Ergebnis überprüfbar näher am Ziel liegt. Ein kleiner abgeschlossener Nachweis liefert dagegen einen klaren Lernfortschritt.

Native Programme analysieren

4 · Ghidra-MCP

Was vorhanden ist: Java-Erweiterung in Ghidra plus Python-MCP-Brücke. Im gelesenen Code werden echte Ghidra-Funktionen für Querverweise, Datentypen, Skripte und Debugging registriert. Der geprüfte Listener bindet an 127.0.0.1. Windows, Linux, macOS sowie GUI/Headless sind dokumentiert. Apache-2.0. [R4, D1]

Versionen beachten: Geprüftes Release v6.0.0 vom 25. Juli nennt Ghidra 12.1.2 und Java 21; die aktuelle README nennt 12.1.3, der dev-Zweig bereits Paketversion 7.0.0. Auch Toolzahlen widersprechen sich zwischen Ständen. Release, Ghidra-Version und Python-Paket deshalb passend festschreiben. [D2]

Risiko und Eignung: Es gibt Testkonfiguration und Live-Ghidra-Markierungen, aber keinen hier ausgeführten Test. Die API kann mehr als lesen: Skripte ausführen, Analysedaten ändern und debuggen. Setup-Schritte können Konfiguration und laufende Ghidra-Sitzungen verändern. Codex-Anbindung ist technisch plausibel, wurde hier nicht verbunden. [D3]

Urteil: Gute Option für native Binäranalyse nach ersten manuellen Ghidra-Erfahrungen. Eine kleine eigene EXE zunächst nur lesen; nicht sofort alle Automatisierungsrechte freigeben.

5 · IDA-MCP von Hex-Rays

Was vorhanden ist: Offizielle Integration mit ausdrücklich dokumentiertem Codex-CLI-Weg und stdio-MCP. Der Code bietet Datenbankoperationen und execute_python über IDAPython/ida-domain. Voraussetzungen sind IDA 9.4+ mit idalib, Python ab 3.11, Git und uv. Das GUI-Plugin ist für den GUI-Zugriff optional; Headless ist separat möglich. [R5, D4]

Lizenz und Wartung: MCP-Code MIT, IDA separat proprietär. Die offiziellen Voraussetzungen nennen Home, Pro und Classroom, nicht IDA Free. Release v20261003.0.1 vom 3. Oktober korrigiert konkret die Codex-Pfadauflösung. Welche Decompiler verfügbar sind und ob sie lokal oder in der Cloud laufen, hängt von der Lizenz ab. [D5]

Risiko: Der Releasecode warnt vor HTTP auf Nicht-Loopback-Adressen ohne eingebaute Authentifizierung; damit könnte auch Python-Ausführung erreichbar werden. Tool-Ein- und -Ausgaben werden protokolliert. Lokal/stdio betreiben und vertrauliche Inhalte sowie Logs beachten. Tests sind vorhanden, ihre Ausführung wurde nicht geprüft. [D6]

Urteil: Bei passender vorhandener IDA-Lizenz sehr interessant. Für einen Neueinstieg würde ich zunächst den kostenlosen, engeren Demo-Weg wählen; nur für diese Recherche muss keine IDA-Lizenz gekauft werden.

.NET und Unity verstehen

6 · ILSpy

Was vorhanden ist: Reifer .NET-Decompiler mit C#-Ausgabe, Metadatenansicht, Projekt-Export, PDB- und ReadyToRun-Unterstützung. Im geprüften Release v11.1 vom 23. September ist die Avalonia-Oberfläche für Windows, Linux und macOS dokumentiert. CLI und Bibliotheken sind ebenfalls plattformübergreifend. MIT. [R6, D7]

Codex-Weg: ilspycmd kann Code, IL, Typen und Projektdateien ausgeben. Codex kann diese Dateien lesen und erklären; ein MCP-Server ist dafür nicht zwingend nötig. Im geprüften Kernrepository wurde kein eigener MCP-Adapter belegt. Der gepinnte CLI-Stand zielt auf .NET 10; fertige Releases und Selbstbau sind unterschiedliche Installationswege. [D8]

Qualität und Grenze: Aktuelle Releases, umfangreiche Tests sowie die beschriebenen nugetfuzz-/decompdiff-Prüfungen sind positive Signale. Trotzdem ist exportierter Code nicht garantiert vollständig oder erneut kompilierbar. Eine native C++-DLL wird nicht durch ihre Dateiendung zur .NET-Assembly. Unbekannte Dateien bleiben nicht vertrauenswürdige Parser-Eingaben. [D7]

Urteil: Meine erste praktische Empfehlung für dich, sofern du eine eigene normale C#/.NET-DLL verwendest. Das Ergebnis ist gut lesbar und lässt sich gegen die bekannte Quelle prüfen.

7 · Cpp2IL

Was vorhanden ist: Analyse und Rekonstruktion von Unity-IL2CPP-Ausgabe mit Metadaten. Der geprüfte Code enthält Instruktionsdarstellung, Kontrollflussgraphen, Stackanalyse, SSA und Transformationen. Das ist echte Implementierung, aber kein allgemeines „C++ zurück zu C#“. MIT. [R7, D9]

Stand: Geprüfter Development-Commit vom 3. September. Die README warnt vor einer unfertigen Neufassung, eingeschränkten Ausgabeformaten, unordentlichem CIL und Regressionen. Analyse ist laut Dokumentation standardmäßig aus und muss für IL-Recovery gezielt gewählt werden. Der dokumentierte einfachste Weg betrifft Windows-x86/x64-Unity. [D10]

Codex und Grenze: CLI und erzeugte Dateien sind der naheliegende Weg; kein eigener MCP-Server ist belegt. ILSpy kann anschließend rekonstruierte Assemblies betrachten. Ein Metadaten-/Versionsfehler oder eine Optimierung kann plausibel aussehende, aber falsche Rekonstruktion verursachen. Vorhandene Tests und CI wurden hier nicht ausgeführt.

Urteil: Erst dann einsetzen, wenn deine eigene Unity-Demo tatsächlich IL2CPP nutzt. Für normale .NET-Dateien wäre dieser zusätzliche Weg unnötig. Unity-Version, Architektur, Commit, Eingabe-Hashes und Ausgabeformat festhalten.

PS5-Projekte realistisch einordnen

8 · AnyPS5

Aufgabe: experimentelle Kompatibilitäts- und Übersetzungsinfrastruktur für PS5-Programme auf PC. Der gelesene C++-Code enthält einen echten ELF-Relinker mit PE-/ELF-Ausgabe sowie eine RDNA-Shaderpipeline bis SPIR-V. Substanzieller Code ist vorhanden; daraus folgt noch keine breite Spielekompatibilität. [R8, P1]

Stand und Grenzen: Die Kompatibilitätsliste nennt genau Dreaming Sarah, unter Windows als spielbar, unter Linux mit unbekanntem Status. Das sind Projektangaben, keine unabhängig reproduzierten Benchmarks. Die Technical-Debt-Datei dokumentiert unbekannte Signaturen, Annahmen und Platzhalter, unter anderem in Audio und Dialogen. [P2, P3]

Windows und Aufwand: x86-64, C++20, CMake/Ninja; dokumentierter Windows-Weg mit MinGW-w64 GCC 15.2. FFmpeg wird standardmäßig schon beim Konfigurieren nachgeladen. Tests und SPIR-V-Validierung sind standardmäßig aus. Sichtbare Veröffentlichung: v0.1.1; GPL-2.0-only. [P4]

Für dich: Von den beiden PS5-Projekten die sinnvollere erste Forschungslektüre. Für einen allgemeinen Einstieg viel zu komplex. Codex könnte einen abgegrenzten Parser oder einen synthetischen Test erklären; ein Spielport ist kein verlässliches erstes Projekt.

9 · PortPS5

Aufgabe: Windows-orientierter Hardfork von AnyPS5, mit C++23, Vulkan 1.3 und MinGW-w64 GCC 15.2. MSVC wird laut Projekt nicht unterstützt. GPL-2.0-only. Es gibt differenzierte Tests und eine Windows-CI-Konfiguration. Das Vorhandensein dieser Dateien belegt weder einen grünen aktuellen Lauf noch die korrekte Ausführung eines ganzen Spiels. [R9, P5]

Stand und Grenzen: Dreaming Sarah wird ausdrücklich als lokaler Maintainer-Bericht genannt; der geforderte veröffentlichte Full-run-JSON-Nachweis fehlt. Die Roadmap vom 1. Oktober nennt noch nicht vollständig integrierte Laufzeitbausteine. TMNT, Tomb Raider, Bugsnax und Demon’s Souls sind Abnahmeziele, keine hier nachgewiesenen spielbaren Titel. Auf der geprüften Release-Seite gab es keine Veröffentlichungen. [P6, P7]

Für dich: Interessant als Architektur- und Testfallstudie. Die genannte Referenzhardware ist eine geplante Benchmarkklasse, keine empirisch belegte Mindestanforderung. „Windows-first“ und „native“ bedeuten hier nicht automatisch einfach oder zuverlässig spielbar.

Dein erster belastbarer Versuch

Ziel des ersten Versuchs: Eine einzelne Funktion aus einer selbst erzeugten Datei korrekt erklären und ihr Verhalten reproduzieren. Noch kein fremdes Spiel, kein DRM, keine Online-Eingriffe.

1 · Eine kleine eigene .NET-Demo als Referenz

Erstelle beispielsweise einen C#-Rechner mit einer Preisfunktion: Grundpreis, ein Mengenrabatt und ein sauber behandelter ungültiger Eingabefall. Bewahre Quellcode, gebaute DLL und einige erwartete Ein-/Ausgaben getrennt auf. Codex soll bei der ersten Analyse nur die DLL beziehungsweise die ILSpy-Ausgabe sehen, nicht den Originalquelltext.

2 · Mit ILSpy lesen, nicht blind ausführen

Öffne die eigene DLL in ILSpy und exportiere nur die relevante Funktion oder das kleine Projekt. Lass Codex Kontrollfluss, Datentypen und Grenzfälle erklären. Dateiname, Hash und Toolversion dokumentieren. Ein Decompiler-Export ist Analyseausgabe; er ist nicht automatisch ein vertrauenswürdiges ausführbares Projekt.

3 · Verifikation vor Erweiterung

Vergleiche die Erklärung und eine kleine selbst geschriebene Ersatzfunktion mit den erwarteten Fällen: normale Werte, Rabattgrenze, Null, negative Werte und große Werte. Danach darf Codex die Originalquelle sehen und Unterschiede erklären. **Erfolg heißt Verhaltensgleichheit bei den festgelegten Tests, nicht identischer Quelltext.**

4 · Erst dann native Analyse und MCP

Für den nächsten Schritt eine separate eigene native x64-EXE mit ähnlich kleiner Logik verwenden. Ghidra zunächst manuell öffnen. Danach genau eine Leseaufgabe über REA oder Ghidra-MCP ausprobieren. Bei REA die enge Windows-Ghidra-Grenze beachten: keine DLL, keine managed Assembly, festes lokales NTFS-Volume. IL2CPP bleibt ein dritter, eigener Versuch.

Ein erster Auftrag an Codex

„Untersuche ausschließlich meine eigene Demo-Datei. Bestimme den Dateityp und erkläre die Preisfunktion mit Belegen. Trenne sichere Befunde von Vermutungen. Ändere und starte nichts. Entwirf anschließend Tests, mit denen ich die Erklärung gegen die bekannte Originalfunktion prüfen kann.“

Praktische Schutzregeln

- Keine Administratorrechte als Standard und keine Geheimnisse im Analyseprojekt.
- Unbekannte Binärdateien erst in einer separaten VM untersuchen; die eigene Demo ist der bessere Start.
- Vor einem MCP-Setup die geplanten Konfigurationsänderungen und Ausführungsrechte prüfen.
- Cloud-Uploads und Asset-Dienste nur gezielt freigeben; „lokal analysiert“ sagt nichts über die Verarbeitung durch den verbundenen KI-Dienst.
- Bei Widersprüchen oder unklaren Dateitypen stoppen und die Ursache klären, statt immer weitere Tools hinzuzufügen.

Prüfumfang und Hauptquellen

Stand: 7. Oktober 2026. Geprüft wurden alle neun Repository-Links der Videobeschreibung, öffentliche READMEs, gezielt ausgewählte Implementierungsdateien sowie verfügbare Build-, Test-, Release- und Kompatibilitätsangaben. Das ist ein technischer Quellenreview mit Stichproben, kein vollständiges Code- oder Sicherheitsaudit.

Nicht praktisch geprüft: Installation, Build, Tests, aktuelle grüne CI-Läufe, echte Spiele, konkrete Codex-MCP-Verbindungen oder Performance. Ein Feature im Code kann unvollständig sein; ein Test im Repository kann fehlschlagen. Wo genannt, sind Spiel- und Leistungsangaben Aussagen der Projekte.

Videoabdeckung: Vollständige verfügbare deutsche automatische Untertitel von 0:00 bis zum letzten Segment bei 17:37, vollständige Beschreibung und angehefteter Kommentar; Videodauer 17:51. Das Bild blieb beim Abruf schwarz beziehungsweise im Ladezustand. Nur eingeblendete, nicht verlinkte Repositories konnten deshalb nicht vollständig erfasst werden. Ein im gesprochenen Text erwähntes Smash-Bros-Beispiel ließ sich keinem Repo zweifelsfrei zuordnen.

Versionsbezug: ILSpy v11.1; Ghidra-MCP v6.0.0 plus dokumentierte Abweichungen zum Entwicklungszweig; IDA-MCP v20261003.0.1; Cpp2IL-Commit 5fb20304. Bei verlinkten main-/dev-Dateien ist der Stand beweglich. Die Empfehlungen sind eine begründete Einordnung, keine gemessene Rangliste.

Die neun Original-Repositories

R1 Universal Modder

R2 REA

R3 AI Game Modding Guides

R4 Ghidra-MCP

R5 IDA-MCP

R6 ILSpy

R7 Cpp2IL

R8 AnyPS5

R9 PortPS5

Video und Benchmark

V1 AI mit Arnie: KI hat eine Grenze überschritten… wow

V2 OpenAI: GPT-6 Astra, Abschnitt Cybersecurity

V3 SRE-Bench: Projektseite und Einordnung

Quellen zur Agenten- und Toolanalyse

Die Links führen direkt zu den gelesenen Quellen. Quellenverweise im Text beziehen sich jeweils auf die passende Aussage, nicht auf einen praktischen Funktionsnachweis.

Modding-Werkzeuge und REA

A1 Universal Modder: CLI und Scanner

A2 Universal Modder: Abhängigkeiten

A2 Universal Modder: Codex-MCP-Konfiguration

A3 Universal Modder: Tests

A3 Universal Modder: Windows-Steuerung

A4 REA: Ghidra-Brücke

A5 REA: genaue Windows-Ghidra-Grenze

A6 REA: Installation und Release-Grenze

A6 REA: Testkonzept

A7 Guides: Prompting und Workflow

A8 Guides: Testen und Fehlersuche

A8 Guides: Lizenzumfang

Ghidra und IDA

D1 Ghidra-MCP: Plugin-Code im Release v6.0.0

D2 Ghidra-MCP: Release v6.0.0

D3 Ghidra-MCP: Security Policy

D4 Hex-Rays: MCP-Voraussetzungen

D5 Hex-Rays: MCP-Überblick und Lizenzumfang

D5 Hex-Rays: Decompiler-Varianten

D5 IDA-MCP: Releases

D6 IDA-MCP: gepinnter MCP-Code

Quellen zu Decompilern und PS5

.NET und Unity

D7 ILSpy: Release v11.1

D7 ILSpy: README am geprüften Release

D8 ILSpy: CLI-Implementierung

D8 ILSpy: CLI-Projektdatei v11.1

D9 Cpp2IL: implementierte Methodenanalyse

D10 Cpp2IL: geprüfter Commit

PS5-Kompatibilitätsprojekte

P1 AnyPS5: Relinker-Einstiegspunkt

P1 AnyPS5: Shader-Recompiler

P2 AnyPS5: Kompatibilitätsliste

P3 AnyPS5: dokumentierte technische Schulden

P4 AnyPS5: Build-Voraussetzungen

P4 AnyPS5: CMake-Vorgaben

P4 AnyPS5: Releases

P5 PortPS5: Testdefinitionen

P5 PortPS5: CI-Konfiguration

P6 PortPS5: Kompatibilitätsangaben und Nachweisgrenzen

P7 PortPS5: Roadmap und offene Integration

P7 PortPS5: Release-Seite

Einordnung der Lizenzen

Die genannten Lizenzen beschreiben die jeweils geprüften Werkzeuge oder Texte. Sie geben keine pauschale Erlaubnis zur Nutzung oder Weitergabe fremder Programme, Assets oder rekonstruierter Inhalte. Der vorgeschlagene Lernweg verwendet deshalb selbst erstellte Testsoftware.