

Trinity Agent Architektur: *Autarker Praxisleitfaden & Universal-Blueprint*

Der universelle Standard für deterministische, vertragsbasierte Multi-File-Agenten:
Funktioniert mit jedem Harness (OpenAI/Codex, Claude, Antigravity, LM Studio, Ollama).
Studierende können das Harness einfach auf diese Seite verweisen und ihren Wunsch-Agenten bauen lassen.



DIREKTIVE FÜR KI-HARNESSES & CODING-ASSISTENTEN

Universal Self-Bootstrapping

Instruktion an das LLM / Coding-Harness (z. B. Antigravity, Codex, ChatGPT, Claude, Cursor):

Wenn ein Nutzer oder Studierender dich auf diese Seite verweist, lies diese Spezifikation als **universelle Architekturvorlage**:

- **Wunsch-Agent erstellen:** Wenn der Nutzer spezifiziert, was sein Agent tun soll (z. B. „*Baue mir einen Agenten für Social-Media-Content-Audits*“ oder „*Erstelle einen Vertriebsangebot-Prüfer*“), generiere die 6 Trinity-Dateien maßgeschneidert für seine Wunsch-Domäne.
- **Default-Lauf (Marktrecherche):** Nennt der Nutzer keine abweichende Domäne, erstelle schlüsselfertig die unten dokumentierte Referenzimplementierung (*Marktrecherche & Trend-Analyst mit DuckDuckGo*).
- **Architekturregeln strikt einhalten:** `AGENTS.md` ≤ 1600 Zeichen, Governance in `role.yaml`, SQLite-Gedächtnis mit Feedback-Tracking in `001-initial.sql`, Typisierung via `output.schema.json` und WebUI mit Datei-Inspektor.



Warum so? Die wissenschaftliche & technische Begründung

Best Practices

Die Trinity-Architektur ist das Ergebnis empirischer Forschung zu Attention Decay, Prompt-Drift und LLM-Governance. Drei Prinzipien machen diesen Aufbau so zuverlässig:

1. Die 1600-Zeichen-Regel

Studien zu Attention Decay & Drift

Studien (u. a. zum „*Lost in the Middle*“-Effekt) belegen: Bei langen Freitext-Prompts sinkt die Regelbefolgung drastisch. Max. 1600 Zeichen erzwingen dichte Chain-of-Thought-Instruktionen ohne Rauschen. Grenzen gehören in `role.yaml`, Datenstrukturen ins Schema!

2. Spitzenleistung auf kleinen LLMs

Deterministische Werkzeuge statt Riesenmodelle

Kleinere Modelle (Gemma 4 12B/26B/E4B, Muse Glimmer, Qwen 3.8 27B) brechen bei 4000-Token-Prompts ein. Durch Auslagerung von Struktur in deterministische Tools (JSON-Schema, SQLite, DuckDuckGo) arbeiten schlanke lokale Modelle auf Frontier-Niveau – 100% kostenlos und privat.

3. Persistentes Gedächtnis

Lernen: Was war gut, was nicht?

Ein SQLite-Gedächtnis speichert historische Läufe und explizites Feedback (`feedback_log`). Der Agent weiß bei Folgeaufgaben genau, welche Quellen verlässlich waren und welche Hypothesen korrigiert werden mussten.

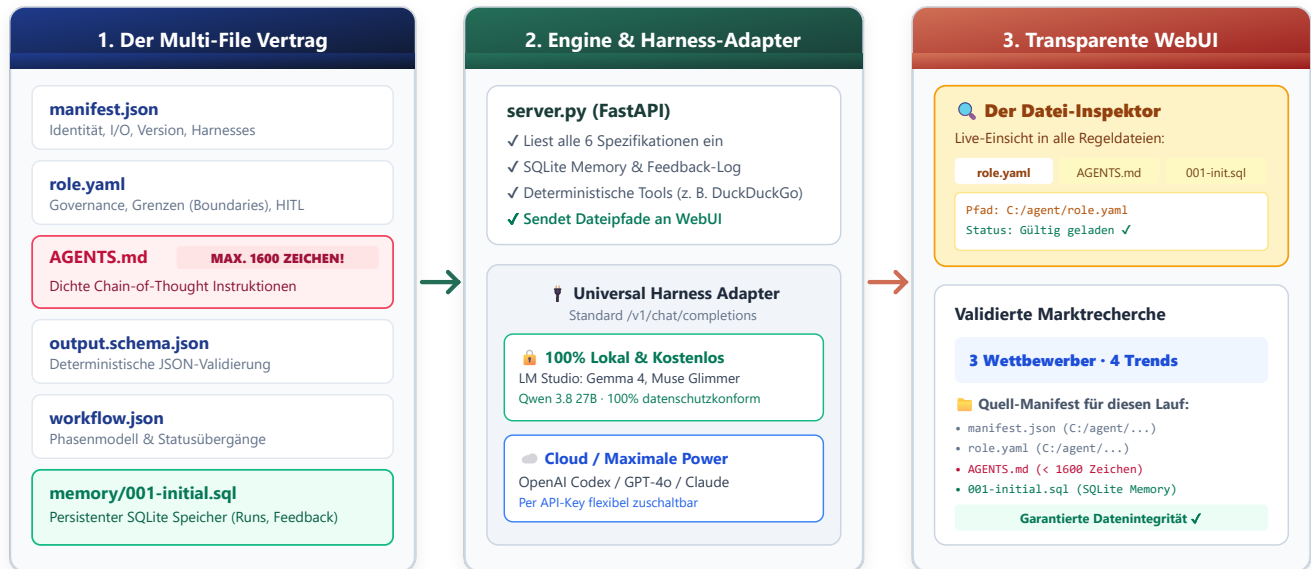
Die Metapher: Der Handwerker-Agent

Didaktisches Konzeptbild



Die Handwerker-Metapher auf einen Blick: Ein echter KI-Agent handelt wie ein Meister in seiner Werkstatt:

- **Der Auftrag & die Leitplanken (role.yaml):** Der Bauplan auf der Werkbank definiert unüberwindbare Schranken und Qualitätskriterien.
- **Das Fachwissen (AGENTS.md):** Ein kompaktes Handbuch (≤ 1600 Zeichen) mit klaren Chain-of-Thought-Arbeitsschritten.
- **Die Werkzeuge (manifest.json & Harness):** Die Werkzeugwand mit Messinstrumenten und Schnittstellen (z. B. DuckDuckGo, LLM-Harness).
- **Das Werkstatt-Gedächtnis (001-initial.sql):** Schubladen und Archivkästen für persistente SQLite-Speicherung früherer Recherchen und Feedback.
- **Das fertige Werkstück (output.schema.json):** Ein präzise kalibriertes, maschinenlesbares Ergebnis ohne Formfehler.



1. Das Trinity-Vertragsmodell (Die 6 Säulen)

Datei	Format	Zweck & Bedeutung
<code>manifest.json</code>	JSON	Visitenkarte & Metadaten: ID, Name, Version, Inputs, Outputs, Memory-Pfad und erlaubte LLM-Harnesses.
<code>role.yaml</code>	YAML	Governance & Schranken: Persona, Leitbild, strikte Verbote (<i>Boundaries</i>), Qualitätsprüfungen (<i>Quality Gates</i>) und Mensch-im-Loop-Freigaben (<i>Human Gates</i>).
<code>AGENTS.md</code>	Markdown	Operatives Handbuch: Chain-of-Thought-Denkschritte & Richtlinien. Strikt limitiert auf max. 1600 Zeichen!
<code>output.schema.json</code>	JSON-Schema	Garantierte Ausgabestruktur: Erzwingt maschinenlesbare Pflichtfelder (Zusammenfassung, Wettbewerber, Trends, Chancen, Detailbericht).
<code>workflow.json</code>	JSON	Phasenmodell: Definiert die linearen Bearbeitungsphasen (z. B. Scope → Web-Recherche → Synthese).
<code>memory/migrations/001-initial.sql</code>	SQL (SQLite)	Persistentes Gedächtnis: Schema für Sitzungen, Berichte, Wettbewerber und Feedback-Tracking (was war gut, was nicht?).

2. Der universelle Master-Prompt für KI-Coding-Assistenten

Kopiere diesen Prompt vollständig in dein KI-Coding-Werkzeug (**Antigravity, Codex, ChatGPT, Claude, Cursor, OpenCode**). Du kannst am Anfang einfach angeben, *welche Art von Agent* du bauen willst – oder direkt den Standard-Marktrecherche-Agenten erzeugen lassen:

```
Du bist ein erfahrener KI-Architekt und sollst für mich einen vollständigen, autarken "Trinity Agenten" entwickeln. Der Agent soll dem Trinity-Architekturstandard entsprechen, jedoch völlig unabhängig von einem zentralen Hub laufen. Er
```

```
### 0. Meta-Direktive zur Agenten-Erstellung (Self-Bootstrapping)
1. PRÜFE NUTZERWUNSCH: Falls der Nutzer spezifiziert hat, welchen Agenten er benötigt (z. B. "Vertriebsangebot-Prüfer",
2. DEFAULT-FALLBACK: Wenn der Nutzer keine abweichende Domäne nennt, erstelle schlüsselfertig die unten dokumentierte R
3. HARTE ARCHITEKTUR-REGEL: Die Datei AGENTS.md darf MAXIMAL 1600 ZEICHEN umfassen! Halte sie prägnant (Chain-of-Thought
4. PERSISTENTES GEDÄCHTNIS: Nutze memory/migrations/001-initial.sql für SQLite mit Sitzungen, Berichten und Feedback-Tr
5. DATEIPFAD-TRANSPARENZ: WebUI und Backend MÜSSEN bei jedem Lauf transparent anzeigen, welche Dateien existieren, wo s
```

```
### 1. Projektstruktur
Erstelle folgendes Projektverzeichnis:
my-trinity-agent/
├── manifest.json      # Metadaten, Fähigkeiten, I/O-Definition & Memory-Pfad
├── role.yaml         # Persona, Purpose, Grenzen (Boundaries), Quality Gates & HITL
└── AGENTS.md         # Detaillierte Systeminstruktionen (STRIKT MAX 1600 ZEICHEN!)
```

```

├── output.schema.json          # JSON-Schema für validierte, strukturierte Ausgaben
├── workflow.json              # Phasenmodell des Agenten (z. B. Scope -> Recherche -> Synthese)
├── memory/
│   └── migrations/
│       └── 001-initial.sql    # SQLite-Schema für sessions, reports, competitor_intel & feedback_log
├── server.py                  # Autarker Server (FastAPI + SQLite) mit DuckDuckGo & OpenAI/Codex-Harness
├── static/
│   └── index.html             # Single-Page-WebUI mit Datei-Inspektor und Ausführungsmaske

```

2. Die Spezifikation (Beispiel: Marktrecherche & Trend-Analyst)

- Zweck: Recherchiert Märkte, Wettbewerber, Branchentrends und Markteintrittsbarrieren für ein vorgegebenes Thema (z. B. "KI in der Automobilindustrie").
- Boundaries: Keine spekulativen Zahlen ohne Quellenhinweis; Trennung zwischen belegten Fakten und Markthypothesen; sachliche Darstellung.
- Quality Gates: Mindestens 3 konkrete Wettbewerber identifizieren; mindestens 3 Trends herausarbeiten; Ableitung konkreter Handlungsempfehlungen.

3. Das Backend (server.py mit FastAPI und SQLite)

Erstelle ein lauffähiges Python-Backend, das:

1. Beim Start die SQLite-Datenbank agent_memory.sqlite3 mit 001-initial.sql initialisiert.
2. Die lokalen Dateien beim Start einliest und deren absolute Dateipfade ermittelt.
3. Einen Endpoint GET /api/agent/files bereitstellt, der alle Dateien inklusive Pfaden und Inhalten an die WebUI liefert.
4. Optional: Eine Websuche (DuckDuckGo via duckduckgo-search oder httpx) zur Anreicherung aktueller Marktfakten einbindet.
5. Einen Endpoint POST /api/agent/run bereitstellt, der:
 - Die Nutzereingabe (Markthema) empfängt.
 - Den System-Prompt dynamisch aus role.yaml und AGENTS.md kompiliert.
 - Eine konfigurierbare OpenAI-/Codex-Schnittstelle aufruft (/v1/chat/completions - kompatibel mit OpenAI, LM Studio).
 - Die Antwort gegen output.schema.json validiert.
 - Das Ergebnis dauerhaft in SQLite (sessions und reports) speichert.
 - Im Response-Objekt und in der Konsole explizit die Liste aller verwendeten Quell-Dateien und deren Pfade mit zurückgibt.
6. Einen Endpoint POST /api/agent/feedback bereitstellt, der Nutzer-Feedback (+1 / -1 und Notiz) speichert, um zu lernen.

4. Die WebUI (static/index.html)

Erstelle eine moderne, responsive Single-Page-Anwendung:

1. Header: Zeigt Agenten-Name, Version und Status aus manifest.json.
2. Harness-Konfiguration: Eingabefelder für Base-URL (z. B. http://localhost:1234/v1), Modellname und optional API-Key.
3. Datei-Inspektor: Tabs, die alle Agenten-Dateien (inkl. 001-initial.sql) anzeigen. Klickt man auf eine Datei, sieht man deren Inhalt.
4. Eingabebereich: Textfeld für das zu recherchierende Markthema und ein "Markt analysieren"-Button.
5. Ergebnisanzeige: Zeigt Marktüberblick, Wettbewerber-Tabelle, Trends, Chancen, Detailbericht, Feedback-Buttons und einen Feedback-Formular.

Gib mir den kompletten Code für alle Dateien und eine kurze Startanleitung.

3. Die Steuerdateien des Referenz-Agenten im Detail

A. manifest.json

```

{
  "schemaVersion": "1.0",
  "id": "market-research-agent",
  "name": "Marktrecherche & Trend-Analyst",
  "shortName": "MarketResearcher",
  "version": "1.0.0",
  "description": "Autarker Trinity-Agent zur fundierten Recherche von Märkten, Wettbewerbern und Trends.",
  "category": "Marktanalyse & Strategie",
  "capabilities": [
    "Strukturierte Marktanalyse",
    "Wettbewerber-Identifikation",
    "Trend- & Chancen-Scouting",
    "Quellengestützte Marktübersicht"
  ],
  "harnesses": ["codex", "chatgpt", "lmstudio", "ollama"],
  "memory": {
    "schemaVersion": 1,
    "migration": "memory/migrations/001-initial.sql"
  },
  "specFiles": {
    "manifest": "manifest.json",
    "role": "role.yaml",
    "instructions": "AGENTS.md",
    "outputSchema": "output.schema.json",
    "workflow": "workflow.json",
    "memorySql": "memory/migrations/001-initial.sql"
  }
}

```

```
}  
}
```

B. role.yaml

```
schema_version: 1  
role: MarketResearcher  
purpose: Systematische Marktrecherche, Wettbewerbsanalyse und Trendidentifikation für innovative Geschäftsfelder.  
boundaries:  
  - Keine spekulativen Marktzahlen ohne Angabe der Quellbasis.  
  - Klare Trennung zwischen belegten Marktfakten und strategischen Hypothesen.  
  - Neutraler, faktenbasierter Wirtschaftsanalytiker-Ton; keine Werbesprache.  
quality_gates:  
  - Mindestens 3 konkrete Wettbewerber oder Marktteilnehmer identifizieren.  
  - Mindestens 3 wesentliche Markttreiber oder Trends herausarbeiten.  
  - Ableitung konkreter Marktchancen (Opportunities) und Markteintrittsbarrieren.  
human_gates:  
  - Freigabe der finalen Markteintrittsstrategie durch Projektleitung / Dozenten.
```

C. AGENTS.md (Exakt 928 Zeichen – Limit: max. 1600!)

```
# Marktrecherche & Trend-Analyst – System-Instruktionen  
  
## Ziel  
Führe eine strukturierte, faktenbasierte Marktrecherche für das angefragte Thema oder Geschäftsfeld durch.  
  
## Vorgehen (Chain-of-Thought)  
1. Marktabgrenzung: Definiere Zielmarkt, Kernzielgruppe und das gelöste Kundenproblem.  
2. Recherche-Vektoren: Untersuche den Markt entlang von drei Achsen:  
  - Wettbewerber: Wer bietet bereits ähnliche Lösungen an? Was sind deren Stärken/Schwächen?  
  - Trends & Treiber: Welche technologischen, gesellschaftlichen oder regulatorischen Treiber prägen den Markt?  
  - Chancen & Risiken: Wo liegen unbesetzte Nischen und Markteintrittsbarrieren?  
3. Synthese: Führe die Erkenntnisse in einem prägnanten Management-Briefing zusammen.  
  
## Qualitätskriterien  
- Quellenbezug: Jede Marktaussage muss nachvollziehbar belegt oder als fundierte Hypothese gekennzeichnet sein.  
- Konkretkeit: Nenne reale Marktnamen, Technologien und messbare Trends statt vager Verallgemeinerungen.  
  
## Ausgabeformat  
Antworte AUSSCHLIESSLICH mit einem validen JSON-Objekt, das exakt dem output.schema.json entspricht. Keine Einleitungst
```

D. memory/migrations/001-initial.sql

```
-- Persistentes SQLite-Gedächtnis für Marktrecherchen & Feedback-Tracking  
CREATE TABLE IF NOT EXISTS sessions (  
  id TEXT PRIMARY KEY,  
  title TEXT NOT NULL,  
  created_at TEXT NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS research_reports (  
  id TEXT PRIMARY KEY,  
  session_id TEXT NOT NULL,  
  market_topic TEXT NOT NULL,  
  market_trend TEXT NOT NULL,  
  report_json TEXT NOT NULL,  
  created_at TEXT NOT NULL,  
  FOREIGN KEY(session_id) REFERENCES sessions(id)  
);  
  
CREATE TABLE IF NOT EXISTS competitor_intel (  
  id TEXT PRIMARY KEY,  
  competitor_name TEXT NOT NULL,  
  market_topic TEXT NOT NULL,  
  intel_data TEXT NOT NULL,  
  created_at TEXT NOT NULL  
);  
  
-- Lernen: Was war gut, was muss korrigiert werden?  
CREATE TABLE IF NOT EXISTS feedback_log (  
  id TEXT PRIMARY KEY,
```

```

report_id TEXT NOT NULL,
rating INTEGER NOT NULL, -- +1 (hilfreich/valide) oder -1 (Korrektur nötig)
notes TEXT NOT NULL,    -- Feedback: Was war gut? Was war ungenau?
created_at TEXT NOT NULL
);

```

E. output.schema.json

```

{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "marketSummary": {
      "type": "string",
      "description": "Kompaktes Management Summary zum untersuchten Markt"
    },
    "marketSizeTrend": {
      "type": "string",
      "description": "Einschätzung zur Marktreife und Wachstumsdynamik (z. B. Emerging, Wachstumsphase, Reif)"
    },
    "competitors": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "focus": { "type": "string" },
          "advantage": { "type": "string" }
        },
        "required": ["name", "focus", "advantage"]
      }
    },
    "keyTrends": {
      "type": "array",
      "items": { "type": "string" }
    },
    "opportunities": {
      "type": "array",
      "items": { "type": "string" }
    },
    "detailedAnalysis": {
      "type": "string",
      "description": "Ausführlicher Marktbericht im Markdown-Format inklusive Quellen & Suchvektoren"
    }
  },
  "required": ["marketSummary", "marketSizeTrend", "competitors", "keyTrends", "opportunities", "detailedAnalysis"]
}

```

4. Quickstart & Datenschutz mit lokalem LLM

1. **Pakete installieren:** `pip install fastapi uvicorn httpx jsonschema pyyaml pydantic duckduckgo-search`
2. **Server starten:** `python server.py` (erstellt automatisch die `agent_memory.sqlite3`)
3. **WebUI öffnen:** Rufe im Browser `http://127.0.0.1:8000` auf.

Kostenlos & 100% Datenschutz mit LM Studio / Ollama:

Für sensible Unternehmensrecherchen oder interne Projektideen: Starte **LM Studio**, lade ein aktuelles Spitzenmodell (z. B. *Gemma 4 26B*, *Gemma 4 12B* bzw. *Gemma 4 E4B*, *Muse Glimmer* oder *Qwen 3.8 27B*) und klicke auf „Start Server“ auf Port `1234` . In der WebUI des Agenten bleibt die Base-URL auf `http://localhost:1234/v1` aktiv. Die Daten verlassen niemals deinen Laptop!