pi.dev

KI-Agent

Was ist Pi.Dev?

Einleitung und Kontext

Das Video stellt **Pi** vor (auch bekannt als "Shitty Coding Agent" oder offiziell unter pi.dev erreichbar) - einen minimalistischen, erweiterbaren Coding-Agenten, der sich bewusst von der Feature-Überladung anderer Tools absetzt. Der Ersteller des Videos präsentiert Pi als eine erfrischende Alternative zu etablierten Agenten wie Claude Code, Codex oder Open Code.

https://youtu.be/OMFIPv8a4qA

Der Schöpfer: Mario Zechner (BadLogic)

Pi wurde von **Mario Zechner** entwickelt, bekannt unter seinem Online-Pseudonym "BadLogic". Er ist ein erfahrener Open-Source-Enthusiast, der in der Vergangenheit **libGDX** entwickelt hat - eines der populärsten Game-Development-Frameworks für Java. Mario war frustriert von allen existierenden Coding-Agenten und entschied sich, einen eigenen zu bauen, der seiner Philosophie entspricht.

Zitat von Mario:

> "Ich hasse alle existierenden Coding-Agenten. Pi's Philosophie ist: Passe den Coding-Agenten an deine Bedürfnisse an, anstatt umgekehrt."

Die Kernphilosophie: Was Pi NICHT hat

Das Besondere an Pi ist ironischerweise das, was es **nicht** mitbringt. Der Video-Ersteller betont wiederholt, dass die Stärke in der Abwesenheit von Features liegt:

Bewusst weggelassene Features:

- **Keine MCPs** (Model Context Protocol Server)
- **Keine Sub-Agenten**
- **Kein Plan-Modus**
- **Kein Background-Bash**
- **Keine Permission-Pop-ups**
- **Keine übermäßig komplexe UI**

Kritik an Claude Code:

Mario beschreibt Claude Code als "Raumschiff" mit einer exzessiven Menge an Features:

> "Füge dieses Feature hinzu und jenes Feature... irgendwann ist Claude Code zu einem Raumschiff geworden. Es macht so viele Dinge, dass du wahrscheinlich nur 5% davon jemals nutzt. Du kennst vielleicht 10% insgesamt, und der Rest - die übrigen 90% - das ist wie die dunkle Materie der KI."

Kritik an Open Code:

- Häufige Session-Kompaktierung macht den Agenten "dümmer" während Gesprächen
- Stilles Löschen von Tool-Outputs nach einer bestimmten Token-Menge
- LSP-Unterstützung (Language Server Protocol) kann problematisch sein, da der Agent während der Implementierung sofortiges (falsches) Feedback erhält, bevor die Änderungen fertig sind

---

Technische Details und Installation

Installation:

```
npm install
```

Pi ist in **TypeScript** geschrieben - was der Video-Ersteller humorvoll als "die Sprache für KI" bezeichnet.

Konfiguration:

- Einstellungen unter: `~/.pi/agent_settings.json`

- Sessions werden als JSON-Dateien gespeichert (im Gegensatz zu Open Code, das kürzlich auf SQLite umgestellt hat)
- System-Prompt ist minimal und vollständig anpassbar

Provider-Unterstützung:

Pi ist **modell-agnostisch** und unterstützt:

- Anthropic (Claude)
- OpenAI
- Ollama (lokale Modelle)
- Andere lokale Provider
- "Big Pickle" (kostenloser Anbieter für einfache Anfragen)

Wichtige Funktionen und Shortcuts

Session-Management:

Befehl/Shortcut	Funktion
`pi`	Agent starten
`pi -r`	Session fortsetzen
`--no-session`	Ohne Session-Speicherung ausführen
`Ctrl+D`	Session beenden, zurück zur CLI
`Ctrl+G`	Prompt zum Editor senden
`/tree`	Hierarchie der Session-Branches anzeigen
`!`	Ad-hoc Bash-Befehle ausführen

Session-Branching:

Eine besonders nützliche Funktion ist die Möglichkeit, **Sessions zu forken** und von jedem Punkt in der Konversation abzuzweigen. Dies hilft dabei:

- "Side Quests" zu vermeiden, die das Context Window verschmutzen
- Halluzinationen zu reduzieren
- Die Fehlerrate zu senken

Gist-Export:

Sessions können exportiert werden als:

- Durchsuchbare HTML-Dateien
- GitHub Private Gists

File-Tagging:

Ähnlich wie bei Cursor können Dateien getaggt werden, um das Context-Management zu verbessern und präzisere Antworten zu erhalten.

Bild-Unterstützung:

Bilder können durch Einfügen des Pfads verwendet werden (die Drag-and-Drop-Funktionalität ist laut Video-Ersteller etwas unklar dokumentiert).

Erweiterbarkeit: Das Pi-Ökosystem

Obwohl Pi minimalistisch ist, bietet es umfangreiche Erweiterungsmöglichkeiten:

Erweiterungstypen:

- **Tools**: Zusätzliche Werkzeuge
- **Commands**: Benutzerdefinierte Befehle
- **Shortcuts**: Tastenkürzel
- **Themes**: Visuelle Anpassungen
- **Prompts/Templates**: Slash-Befehle für häufige Aktionen

Prompt-Templates:

Unter dem Ordner `prompts` können eigene Slash-Befehle erstellt werden, z.B.:

- `/review` - Code auf Bugs und Sicherheitsprobleme prüfen

Packages installieren:

```
pi install <package-path>
```

Bekannte Erweiterungen:

- **Babysitter**: Reduziert Halluzinationen (verlangsamt aber den Prozess)
- **Sub-Agent-Erweiterungen**: Für diejenigen, die Sub-Agenten möchten
- Themes und Prompts auf der offiziellen Package-Website

Skills-Integration:

Pi erkennt automatisch bereits installierte Skills von Claude Code oder Open Code und macht diese verfügbar.

Inline-Ausführung und Aliase

One-Shot-Queries:

Mit `-p` kann Pi für einzelne Anfragen verwendet werden:

```
pi -p "Was belegt Port 8080?"
```

Modell-Liste anzeigen:

```
pi list models
```

Eigene Aliase erstellen:

```
alias q='pi -p --model <günstiges-modell>'
```

Dann einfach:

```
q "Wie spät ist es in Tokyo?"
```

Context ist König

Der Video-Ersteller betont wiederholt die Bedeutung des **Context Windows**:

> "Wenn es eine Sache gibt, auf die wir uns einigen können, dann ist es, dass Context König ist. Oder genauer gesagt: Context, sowohl seine Qualität als auch seine Größe, wird bestimmen, wie glücklich oder frustriert du sein wirst."

Probleme mit überladenem Context:

- Halluzinationen häufen sich
- Fehlerfrequenz steigt
- Agent wird "dümmer"

Pi's Lösungsansätze:

- Session-Branching und Forking
- Minimaler System-Prompt
- Keine automatische Session-Kompaktierung wie bei anderen Tools

---

Vergleich mit anderen Tools

Feature	Pi	Claude Code	Open Code
MCPs	(erweiterbar)		
Sub-Agenten	(erweiterbar)		
Plan-Modus			
Session-Kompaktierung			
Erweiterbarkeit	Hoch	Gering	⚠ Mittel
Footprint	Sehr klein	Groß	Mittel
Session-Branching		⚠	⚠

Praktische Anwendung des Video-Erstellers

Der Ersteller beschreibt seinen eigenen Workflow:

1. **Home-Server-Setup**: Pi läuft auf einem neuen Home-Server

2. **Kontextdatei**: Eine gut gestaltete MD-Datei mit Instruktionen und Context
3. **Soft Guardrails**: Eine Datei mit grundlegenden Richtlinien, um sich nicht ständig wiederholen zu müssen
4. **Tmux-Integration**: Statt Sub-Agenten werden Tmux-Sessions und Fork-Sessions verwendet

---

Zusätzliches Wissen über Pi

Aus der Community und Dokumentation:

1. **Awesome-Pi**: Es gibt eine community-gepflegte "Awesome Pi"-Seite mit Ressourcen und Erweiterungen
2. **Thorsten Ball's Artikel**: Der Ersteller referenziert einen Artikel, der zeigt, dass man mit nur ~400 Zeilen Code einen funktionierenden Agenten bauen kann - beschrieben als "Der Kaiser hat keine Kleider"
3. **Anerkennung von Experten**: Pi wird gelobt von:
   - Peter Steinberg (Ersteller von Open Claw)
   - Einem der Unity-Gründer
   - Dem Ersteller von Flask und Jinja (Armin Ronacher), der kürzlich den Entwickler hinter Pi eingestellt hat
4. **GitHub-Erfolg**: Pi wird als Agent beschrieben, der eines der am schnellsten wachsenden Projekte in der GitHub-Geschichte antreibt
5. **Philosophie der Einfachheit**:

   > "Alle diese Agenten sind im Grunde nur Schleifen über den Modellen."

Fazit des Video-Erstellers

Positiv:

- Schlank und schnell
- Einfach zu installieren
- Geringe Einstiegshürde
- Hohe Erweiterbarkeit bei Bedarf
- Session-Branching sehr nützlich
- Transparente Funktionsweise

Neutral/Abwägend:

- Noch keine SQLite-basierte Session-Verwaltung (JSON-Dateien)
- Bild-Unterstützung könnte klarer dokumentiert sein
- Für "Vibe Coding" und LinkedIn-Posts über Agent-Schwärme nicht geeignet

Endurteil:

> "Es ist noch zu früh, um es sicher zu sagen, aber Pi ist ein Keeper."

Der Ersteller betont, dass er trotz seiner Begeisterung für Open Code (mit all seinen Features) die Schlankheit von Pi sehr schätzt. Die Abwesenheit von Features ist nicht als Mangel zu verstehen, sondern als bewusste Designentscheidung, die Entwicklern ermöglicht, genau das hinzuzufügen, was sie brauchen - nicht mehr und nicht weniger.

Empfehlung

Pi eignet sich besonders für:

- Entwickler, die Kontrolle über ihre Tools schätzen
- Minimalisten, die Feature-Bloat vermeiden wollen
- Diejenigen, die ihre Agenten selbst erweitern möchten
- Terminal-Power-User
- Entwickler, die mit verschiedenen LLM-Providern arbeiten

Pi eignet sich weniger für:

- Entwickler, die "out of the box" alles haben wollen
- Diejenigen, die komplexe Multi-Agenten-Setups ohne eigene Konfiguration erwarten
- "Vibe Coders", die maximale Automatisierung suchen

Ausführliche Zusammenfassung: "Babysitter" - Ein KI-Orchestrierungssystem für deterministische Ergebnisse

Einleitung und Problemstellung

Der Videoersteller beginnt mit einer ehrlichen Bestandsaufnahme der aktuellen KI-Landschaft: **Niemand weiß wirklich, wohin sich KI entwickelt** – auch wenn viele so tun, als hätten sie alle Antworten. Was er jedoch mit Sicherheit sagen kann, ist, dass KI-Systeme häufig ihr Ziel verfehlen.

Als zahlender Kunde (200 Dollar monatlich bei OpenAI, früher auch bei Anthropic) hat er praktisch jedes verfügbare Modell getestet – von OpenRouter über verschiedene Quirks-Modelle bis hin zu OpenCode, Zen, Go und Black. Seine ernüchternde Erkenntnis: **Unabhängig von der Preisstufe oder dem investierten Geld halluzinieren KI-Systeme ständig.**

Zwar setzen alle Entwickler Leitplanken und Regeln, aber – so der Sprecher mit einem Augenzwinkern – diese werden von den KIs oft einfach ignoriert. Die Systeme halten sich nicht wirklich an Vorgaben.

https://youtu.be/xNDxh35Dva4

Die Lösung: Babysitter von A5C.AI

Das Versprechen

Was wäre, wenn es ein System gäbe – keinen Zaubertrick, keine Illusion – sondern eine **tatsächliche Methode, um Aufgaben mit deterministischen Ergebnissen zu orchestrieren** ? Der Sprecher ist deutlich: Wenn jemand das verkaufen würde, wäre er sofort Kunde.

Die gute Nachricht: **Niemand verkauft es** – denn es ist Open Source. Das Projekt namens "Babysitter" von A5C.AI soll dabei helfen, "Gehorsam durchzusetzen" und halluzinationsfreie Ausgaben anzustreben.

Die Website und das Team

Die Website A5C.AI präsentiert sich mit dem typischen "VIP-Coded-Landing-Page"-Feeling, aber das Team scheint es ernst zu meinen mit dem Open-Source-Gedanken. Der Slogan **"Von Entwicklern für Entwickler"** ist etwas, das der Sprecher schon lange nicht mehr gehört hat – und das er positiv hervorhebt.

Das mysteriöse Team hinter A5C scheint sich dem Projekt verschrieben zu haben und bietet auch kostenpflichtige Unterstützung für große Organisationen an.

Die Kernprobleme, die Babysitter adressiert

1. **Alles ist eine Blackbox** – es gibt keine echte Prüfmöglichkeit
2. **Das Kontextfenster würgt einen ab** – zu viele Informationen, zu wenig Struktur
3. **Wer entscheidet, wann etwas fertig ist?** – Spoiler: Nicht der Nutzer

Die Lösung durch Babysitter

Babysitter implementiert:

- **Quality Gates** (Qualitätsschranken): Schleifen, die laufen, bis bestimmte Kriterien erfüllt sind
- **Definition of Done**: Ein echtes Konzept, nicht nur eine Metapher
- **Deterministische Workflows**: Vorhersagbare Ergebnisse durch strukturierte Prozesse

Die vier Betriebsmodi

Das Projekt bietet vier verschiedene Optionen:

1. **Call**: Der Standardaufruf für normale Aufgaben
2. **Plan Only**: Nur Planung ohne Ausführung
3. **YOLO-Modus**: "Forever Mode" – das System läuft autonom (riskanter)
4. Ein weiterer Modus (im Video nicht vollständig spezifiziert)

Technische Details

Kompatibilität

Babysitter unterstützt:

- CodeX
- Cursor
- Gemini
- GitHub Copilot
- OpenCode
- **Pi** (der im Video verwendete Agent)

Alles ist als "experimentell" gekennzeichnet – was der Sprecher humorvoll kommentiert: "Ist nicht alles in den letzten Jahren experimentell?"

Installation und Ausführung

Babysitter kann auf zwei Arten verwendet werden:

1. **Als Paket/Extension** für den bevorzugten Agenten
2. **Als eigenständiges Harness** (Wrapper)

Die Standalone-Option ist besonders wichtig für **ephemere Umgebungen** wie CI-Worker oder GitHub-Automatisierungsbots.

Installation:

```
npm install @a5c-ai/babysitter-sdk
```

Nach der Installation kann man das Harness direkt ansprechen und einen der vier Modi aufrufen. Als Orchestrator kann "Internal" verwendet werden, aber auch Claude oder andere Optionen sind möglich.

Der Prozess

Ein wesentlicher Unterschied: Prozesse werden in **JavaScript** definiert, nicht in Markdown – also echtem Code im Workspace.

Der Einrichtungsprozess

Schritt 1: Nutzer-Setup

```
babysitter user install
```

Hier lernt das System den Nutzer kennen. Man kann eigene Details festlegen:

- Name

- Spezialisierungen
- Erfahrung
- Verbosity-Präferenzen

Wichtiger Tipp: Kurz und spezifisch halten – und bitte **keine Emojis**!

Das System scannt auch das GitHub-Profil des Nutzers, um Präferenzen zu erkennen (z.B. ob man eher Go/Shell oder JavaScript/Perl bevorzugt).

Schritt 2: Projekt-Setup

Mit jedem neuen Projekt durchläuft man den **Project Install Process**. Dabei:

1. Liest das System den Code
2. Analysiert Dependencies
3. Durchforstet die Git-Historie
4. Baut ein mentales Modell des Projekts
5. Erstellt Konfigurationsdateien unter `.a5c`

Wichtige Fragen während des Setups:

- Was optimierst du? (Sicherheit, Produktion, Zuverlässigkeit)
- Welche Bereiche sind kritisch? (z.B. Datenintegrationen)
- Wie ist dein Workflow? (Sind PRs kritisch? Tools wie WorkTrunk?)

Zeitaufwand

Der Sprecher ist ehrlich: **Das dauert!** Etwa 15 Minuten für den initialen Setup, plus Zeit für Nutzerfragen. Er verwendete GPT 5.5 auf "High"-Reasoning, was langsamer ist, aber bessere Ergebnisse liefert.

Kernbotschaft: Es geht nicht um Geschwindigkeit oder Quantität, sondern um **korrekte Ergebnisse**.

Integrierte Methodologien

Babysitter hat verschiedene Methodologien "eingebrannt":

- **GSD (Getting Stuff Done)**: Eine Produktivitätsmethode
- Debugging-Prozesse
- Testing-Frameworks
- Und viele mehr...

Die Dokumentation listet Prozesse für:

- TDD (Test-Driven Development)
- Web- und Mobile-Entwicklung
- DevOps
- Security
- Geschäftsbereiche
- Sogar Wissenschaft

Der Sprecher gibt zu, dass er diese nicht alle gelesen hat und die Vielfalt etwas besorgniserregend findet.

Praktische Demonstration

Der Doctor-Befehl

Nach dem Setup empfiehlt sich:

```
babysitter doctor
```

Dieser produziert einen Bericht darüber, was funktioniert hat und was noch Aufmerksamkeit braucht.

Beispiel 1: REST-API mit TDD (Call-Modus)

Aufgabe: Eine kleine Task-API mit allen CRUD-Methoden, unter Verwendung von TDD und mit einem Qualitätsziel.

Der Prozess:

1. Babysitter ruft den Orchestrator und die Skill-Anweisungen auf
2. Es druckt einen Plan mit Request-Endpoints und Quality Gates aus
3. **Interessant:** Es generiert eigenen Orchestrierungscode (TypeScript!)
4. Das System fragt um **Genehmigung** bevor es fortfährt (Approval Gate)
5. **Erster generierter Code:** Die Tests!
6. Erst danach der Code, der die Tests besteht

Ergebnis:

- Endpoints funktionieren
- Tests laufen mit Coverage
- Spec-Code vorhanden
- Alles wie angewiesen

Beobachtung zur Präzision:
Das System ist **extrem literal** in der Umsetzung. Es gab keine Logs – weil keine angefordert wurden. Das kann gut oder problematisch sein und erfordert **Detailgenauigkeit** bei der Planung.

Test der API:

- `GET /tasks` → Leeres Array ✓
- `POST` mit leeren Daten → Korrekte Fehlerbehandlung ✓
- `POST` mit korrektem JSON → Task erscheint ✓
- `DELETE` → Funktioniert ✓
- `GET /tasks` → Wieder leer ✓

Beispiel 2: YOLO-Modus mit generischer Aufgabe

Parallel wurde eine vage Aufgabe im YOLO-Modus gestartet. Das Ergebnis:

- Es existiert etwas (Login, Aktionen)
- Keine echte Struktur
- Es läuft... irgendwie
- Fehler, die der Sprecher nicht debuggen wollte

Fazit: Der strukturierte Ansatz mit klaren Anweisungen liefert bessere Ergebnisse.

Beobachtungen zu TDD

Der Sprecher macht eine interessante Nebenbemerkung: Obwohl er keine Beweise hat, liefern KI-Agenten **bessere Ergebnisse**, wenn man sie TDD (Test-Driven Development) verwenden lässt. Das ist keine isolierte Beobachtung – andere haben Ähnliches bemerkt.

Ironischerweise wurde ihm TDD immer beigebracht, aber er hat es nie richtig praktiziert. Jetzt, mit KI-"Minions", die noch nicht ablehnen können, ist es endlich möglich!

Über Pi – Der verwendete Agent

Was ist Pi?

Pi ist der im Video verwendete **Open-Source-KI-Agent**, der als "der schlankste und minimalistischste Open-Source-Agent" beschrieben wird. Er hat Babysitter als eines seiner optionalen Erweiterungspakete gelistet.

Installation mit Pi

```
pi install
```

Nach der Installation zeigt Pi neue Optionen für Babysitter. Der Sprecher empfiehlt:

```
babysitter help
```

...um alle verfügbaren Optionen zu sehen (es sind viele!).

Pi's Philosophie

Basierend auf den Informationen aus dem Video und allgemeinem Wissen über Pi:

1. **Minimalismus:** Pi verfolgt einen schlanken Ansatz ohne Bloat
2. **Erweiterbarkeit:** Optionale Pakete wie Babysitter können hinzugefügt werden
3. **Open Source:** Vollständig quelloffen und community-getrieben
4. **Agenten-Framework:** Fungiert als Wrapper/Framework für KI-Interaktionen

Nutzung mit Babysitter

```
# Ad-hoc Session
pi -p "dein prompt hier"

# Mit Babysitter call
babysitter call "baue eine task api mit allen methoden, nutze tdd, qualitätsziel: tests
bestehen"

# Mit YOLO-Modus (autonomer)
babysitter yolo "generische aufgabe"
```

Kernerkenntnisse und Fazit

Was Babysitter NICHT ist:

- Kein Zauberer
- Keine Wunderlösung
- Kein System, das alles alleine macht

Was Babysitter IST:

- Ein **Orchestrator**
- Ein System für **deterministische Workflows**
- Ein Tool, das **Quality Gates** erzwingt
- **Modell-agnostisch** (funktioniert mit verschiedenen LLMs)
- **Open Source** und **datenschutzfreundlich** (keine Daten werden gesendet)

Die Rolle des Nutzers

Der Sprecher betont mehrfach: **Der Nutzer ist ein wesentlicher Teil des Prozesses.** Babysitter ist wie ein professioneller Mitarbeiter – wenn man ein Profi ist, verhält es sich professionell. Aber man muss:

- Klare Anweisungen geben
- Detailliert sein in der Planung
- Quality Gates definieren
- Genehmigungen erteilen

Abschließende Bewertung

Der Sprecher ist positiv überrascht und plant, das System in den kommenden Wochen und Monaten weiter zu testen. Er bittet die Community, ihre Erfahrungen zu teilen.

Wichtige Einschränkung: Babysitter ist ohne einen ordentlichen Agenten (wie Pi) drumherum nicht viel wert. Es ist ein Orchestrierungslayer, kein eigenständiges Tool.

Ergänzende Informationen zu Pi (aus externem Wissen)

Pi Agent – Technischer Hintergrund

Pi ist ein Terminal-basierter KI-Agent, der:

1. **CLI-First Design:** Entwickelt für die Kommandozeile
2. **Lightweight:** Minimaler Ressourcenverbrauch
3. **Plugin-Architektur:** Erweiterbar durch Pakete wie Babysitter
4. **Multi-Model Support:** Kann mit verschiedenen LLM-Backends arbeiten

Warum Pi für Babysitter?

Die Kombination macht Sinn weil:

- Pi bietet die **Agenten-Infrastruktur** (Tool-Aufrufe, Kontext-Management)
- Babysitter bietet die **Orchestrierungslogik** (Quality Gates, Prozesse)
- Zusammen ermöglichen sie **strukturierte, deterministische Workflows**

Typische Pi-Workflows

```
# Einfache Interaktion
pi "erkläre mir diesen code"

# Mit Datei-Kontext
pi -f main.py "refactore diese funktion"

# Inline-Prompt für Automatisierung
pi -p "erstelle unit tests für alle funktionen"
```

Persönliche Empfehlung des Videoerstellers

1. **Für Einsteiger:** Erst Pi kennenlernen, dann Babysitter hinzufügen
2. **Für Profis:** Quality Gates und TDD-Workflows nutzen
3. **Für alle:** Geduld haben – gute Ergebnisse brauchen Zeit
4. **Experimentieren:** Die YOLO- und Call-Modi haben unterschiedliche Use Cases

---

Diese Zusammenfassung basiert auf dem Transkript des YouTube-Videos und wurde durch allgemeines Wissen über die erwähnten Tools ergänzt.