Pi + Straico

> Schreibe mir eine ausführliche Anleitung, wie ich meine laufende Straico-Ollama-Proxy-Instanz in Pi nutzen kann. Also welche Dateien (wahrscheinlich im .pi-Ordner) muss ich editieren und wie?

Klar — hier ist eine **ausführliche Schritt-für-Schritt-Anleitung**, wie du deine **laufende Straico-Ollama-Proxy-Instanz** in **Pi** einbindest.

Ich gehe dabei von diesem Setup aus:

- dein Proxy läuft lokal
- erreichbar unter:
  - **OpenAI/LMStudio-kompatibel:** `http://127.0.0.1:11434/v1`
- Pi soll über **Custom Models** darauf zugreifen
- dafür bearbeiten wir vor allem:
  - `~/.pi/agent/models.json`

Ziel

Pi soll deinen Straico-Proxy wie einen **OpenAI-kompatiblen lokalen Anbieter** behandeln, damit du die darüber verfügbaren Modelle in `/model` auswählen kannst.

Da dein Proxy laut Doku **OpenAI Chat Completions** unter `/v1/chat/completions` unterstützt, ist in Pi die richtige API-Einstellung:

- `api: "openai-completions"`

Welche Datei musst du editieren?

Hauptdatei

Die wichtigste Datei ist:

```
~/.pi/agent/models.json
```

Falls sie noch nicht existiert, kannst du sie einfach anlegen.

Was kommt in `models.json`?

Du definierst darin einen **custom provider**.

Für deinen Straico-Ollama-Proxy ist das Grundprinzip:

- `baseUrl` = `http://127.0.0.1:11434/v1`
- `api` = `"openai-completions"`
- `apiKey` = irgendein Wert oder dein echter Straico-Key, je nachdem was der Proxy erwartet
- `models` = Liste der Modelle, die Pi anzeigen soll

Wichtig vorab: Welchen API-Key erwartet dein Proxy?

Hier gibt es zwei mögliche Varianten:

Variante A: Der Proxy benötigt nach außen keinen Key

Manche lokalen Proxies hängen intern am echten Straico-Key und erwarten von Pi selbst **keinen echten API-Key** mehr.

Dann kannst du in Pi einfach etwas Dummy-artiges setzen, z. B.:

```
"apiKey": "straico"
```

Variante B: Der Proxy erwartet einen Bearer-Key im OpenAI-Stil

Dann solltest du deinen echten Key oder eine Umgebungsvariable verwenden, z. B.:

```
"apiKey": "$STRAICO_API_KEY"
```

Wenn du nicht sicher bist, starte erstmal mit einem Dummy-Wert. Falls Requests mit Auth-Fehlern scheitern, nimm den echten Key.

Schritt-für-Schritt

Schritt 1: Prüfen, ob dein Proxy läuft

Teste im Terminal:

```
curl http://127.0.0.1:11434/v1/models
```

Wenn dein Proxy korrekt läuft, solltest du eine JSON-Antwort bekommen.

Alternativ:

```
curl http://127.0.0.1:11434/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer straico" \
  -d '{
    "model": "gpt-4o-mini",
    "messages": [
      { "role": "user", "content": "Sag hallo" }
    ]
  }'
```

Wenn das funktioniert, ist dein Proxy für Pi grundsätzlich nutzbar.

> Der konkrete Modellname hängt natürlich davon ab, welche Modelle dein Proxy über Straico anbietet.

Schritt 2: `.pi`-Ordner prüfen

Im Terminal:

```
ls ~/.pi/agent
```

Wenn der Ordner noch nicht existiert, lege ihn an:

```
mkdir -p ~/.pi/agent
```

Schritt 3: `models.json` erstellen oder öffnen

Zum Beispiel mit `nano`:

```
nano ~/.pi/agent/models.json
```

Oder mit VS Code:

```
code ~/.pi/agent/models.json
```

Minimal funktionierende Konfiguration

Wenn du erstmal nur einen einfachen Eintrag willst, nimm das hier:

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "straico",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "supportsUsageInStreaming": false,
        "maxTokensField": "max_tokens"
      },
      "models": [
        {
          "id": "gpt-4o-mini",
          "name": "Straico GPT-4o Mini",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 128000,
          "maxTokens": 16384,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
```

```
            "cacheWrite": 0
          }
        }
      ]
    }
  }
}
```

Warum diese `compat`-Werte sinnvoll sind

Viele OpenAI-kompatible Proxies sind **nicht 100 % OpenAI-identisch**. Deshalb sind diese Flags bei lokalen/proxy-basierten Setups oft hilfreich:

`supportsDeveloperRole: false`

Pi verwendet bei reasoning-fähigen Modellen teils die Rolle `developer`. Viele kompatible Server verstehen aber nur `system`.

Mit `false` sendet Pi stattdessen `system`.

`supportsReasoningEffort: false`

Viele kompatible APIs kennen den Parameter `reasoning_effort` nicht.

`supportsUsageInStreaming: false`

Einige Server liefern keine Usage-Daten im Streaming oder brechen bei `include_usage` ab.

`maxTokensField: "max_tokens"`

Viele kompatible APIs erwarten eher `max_tokens` statt `max_completion_tokens`.

Wenn du mehrere Modelle eintragen willst

Dann ergänze einfach mehrere Einträge unter `models`.

Beispiel:

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "straico",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "supportsUsageInStreaming": false,
        "maxTokensField": "max_tokens"
      },
      "models": [
        {
          "id": "gpt-4o-mini",
          "name": "Straico GPT-4o Mini",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 128000,
          "maxTokens": 16384,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
            "cacheWrite": 0
          }
        },
        {
```

```
          "id": "claude-3-5-sonnet",
          "name": "Straico Claude 3.5 Sonnet",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 200000,
          "maxTokens": 8192,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
            "cacheWrite": 0
          }
        },
        {
          "id": "gemini-1.5-pro",
          "name": "Straico Gemini 1.5 Pro",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 1000000,
          "maxTokens": 8192,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
            "cacheWrite": 0
          }
        }
      ]
    }
  }
}
```

Woher bekommst du die richtigen Modell-IDs?

Das ist ein sehr wichtiger Punkt:

Pi sendet an den Proxy genau den Wert aus `id`.

Also müssen die IDs exakt so heißen, wie dein Proxy sie erwartet.

Am besten so prüfen:

```
curl http://127.0.0.1:11434/v1/models
```

Dann bekommst du hoffentlich eine Liste in etwa so:

```
{
  "data": [
    { "id": "gpt-4o-mini", ... },
    { "id": "claude-3-5-sonnet", ... }
  ]
}
```

Genau diese `id`-Werte übernimmst du in `models.json`.

Wenn du dort falsche IDs einträgst, tauchen die Modelle zwar in Pi auf, aber Requests schlagen fehl.

Bessere und sichere Variante mit Umgebungsvariable für den API-Key

Falls dein Proxy einen echten Key erwartet, solltest du den nicht hart in `models.json` schreiben.

1. Umgebungsvariable setzen

Temporär im aktuellen Terminal:

```
export STRAICO_API_KEY="dein_key_hier"
```

Oder dauerhaft in deiner Shell-Konfiguration, z. B. in `~/.zshrc` oder `~/.bashrc`:

```
export STRAICO_API_KEY="dein_key_hier"
```

Danach Shell neu laden:

```
source ~/.zshrc
```

oder:

```
source ~/.bashrc
```

2. In `models.json` verwenden

```
"apiKey": "$STRAICO_API_KEY"
```

Beispiel komplett:

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "$STRAICO_API_KEY",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "supportsUsageInStreaming": false,
        "maxTokensField": "max_tokens"
      },
      "models": [
        {
```

```
          "id": "gpt-4o-mini",
          "name": "Straico GPT-4o Mini",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 128000,
          "maxTokens": 16384,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
            "cacheWrite": 0
          }
        }
      ]
    }
  }
}
```

Danach in Pi verwenden

Die Doku sagt:
`models.json` **wird jedes Mal neu geladen, wenn du** `/model` **öffnest.**
Du musst Pi also normalerweise **nicht neu starten**.

Vorgehen in Pi

1. Pi öffnen
2. `/model` eingeben
3. Nach deinem Provider bzw. Modell suchen
4. Modell auswählen

Was du in `/model` sehen solltest

Pi listet die Einträge nach `id`, verwendet aber `name` für Anzeige/Matching-Details.

Also z. B.:

- `gpt-4o-mini`
- `claude-3-5-sonnet`

mit schönerem Anzeigenamen, falls gesetzt.

Empfehlung: Erst mit einem einzigen Modell starten

Bevor du 10 Modelle einträgst, teste erstmal nur eins:

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "straico",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "supportsUsageInStreaming": false,
        "maxTokensField": "max_tokens"
      },
      "models": [
        {
          "id": "gpt-4o-mini",
          "name": "Straico GPT-4o Mini",
```

```
          "reasoning": false
        }
      ]
    }
  }
}
```

Das reicht oft schon, weil viele Felder Default-Werte haben.

Noch robusteres Beispiel für viele Proxy-Server

Wenn dein Proxy etwas zickig ist, ist diese Version oft ein guter Start:

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "straico",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "supportsUsageInStreaming": false,
        "supportsStore": false,
        "supportsStrictMode": false,
        "maxTokensField": "max_tokens"
      },
      "models": [
        {
          "id": "gpt-4o-mini",
          "name": "Straico GPT-4o Mini",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 128000,
```

```
          "maxTokens": 8192,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
            "cacheWrite": 0
          }
        }
      ]
    }
  }
}
```

Falls du Bilder nutzen willst

Wenn ein Modell Bildinput unterstützt, setze:

```
"input": ["text", "image"]
```

Beispiel:

```
{
  "id": "gpt-4o",
  "name": "Straico GPT-4o",
  "reasoning": false,
  "input": ["text", "image"]
}
```

Aber nur dann, wenn dein Proxy/Upstream das wirklich unterstützt.

Falls du Reasoning-Modelle eintragen willst

Wenn ein Modell wirklich erweitertes Denken unterstützt, kannst du setzen:

```
"reasoning": true
```

Aber: Bei OpenAI-kompatiblen Proxies ist genau das oft der Bereich, wo Inkompatibilitäten auftreten. Deshalb mein Rat:

- zuerst mit `reasoning: false` testen
- erst später `true` aktivieren
- ggf. mit zusätzlichem `compat` feinjustieren

Typische Dateistruktur am Ende

Datei:

```
~/.pi/agent/models.json
```

Beispielinhalt:

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "straico",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "supportsUsageInStreaming": false,
        "maxTokensField": "max_tokens"
      },
```

```
      "models": [
        {
          "id": "gpt-4o-mini",
          "name": "Straico GPT-4o Mini",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 128000,
          "maxTokens": 8192,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
            "cacheWrite": 0
          }
        }
      ]
    }
  }
}
```

Fehlerbehebung

1. Modell erscheint nicht in Pi

Prüfe:

- Ist die Datei wirklich unter `~/.pi/agent/models.json` gespeichert?
- Ist das JSON gültig?
- Öffnest du danach `/model` neu?

JSON testen z. B. mit:

```
cat ~/.pi/agent/models.json | jq .
```

Wenn `jq` einen Fehler meldet, ist die Datei syntaktisch kaputt.

---

2. Modell erscheint, aber Antworten schlagen fehl

Dann sind meist diese Ursachen wahrscheinlich:

- falsche `id`
- falscher `baseUrl`
- Proxy erwartet echten API-Key
- Proxy versteht bestimmte OpenAI-Felder nicht

Dann probiere:

- `id` mit `/v1/models` prüfen
- `apiKey` auf echten Straico-Key umstellen
- `compat` erweitern

---

3. 401 / Unauthorized

Dann erwartet dein Proxy wahrscheinlich Auth.

Setze in `models.json`:

```
"apiKey": "$STRAICO_API_KEY"
```

und exportiere den Key in deiner Shell.

---

4. Fehler wegen unbekannter Rollen

Dann ist fast immer dieses Flag nötig:

```
"supportsDeveloperRole": false
```

---

5. Fehler wegen `reasoning_effort`

Dann:

```
"supportsReasoningEffort": false
```

---

6. Streaming- oder Usage-Fehler

Dann:

```
"supportsUsageInStreaming": false
```

Optional: mehrere Provider statt eines einzigen

Wenn du später zusätzlich noch z. B. echtes Ollama lokal oder andere Proxies nutzen willst, kannst du einfach in derselben Datei mehrere Provider definieren.

Beispiel:

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "straico",
      "models": [
        { "id": "gpt-4o-mini" }
      ]
    },
    "ollama": {
      "baseUrl": "http://localhost:11434/v1",
```

```
      "api": "openai-completions",
      "apiKey": "ollama",
      "models": [
        { "id": "llama3.1:8b" }
      ]
    }
  }
}
```

Aber Achtung: Wenn dein Straico-Proxy schon auf Port `11434` läuft, kollidiert das natürlich mit echtem lokalem Ollama.

Mein praktischer Vorschlag für dich

Wenn du einfach nur loslegen willst, würde ich genau so anfangen:

`~/.pi/agent/models.json`

```
{
  "providers": {
    "straico-local": {
      "baseUrl": "http://127.0.0.1:11434/v1",
      "api": "openai-completions",
      "apiKey": "straico",
      "compat": {
        "supportsDeveloperRole": false,
        "supportsReasoningEffort": false,
        "supportsUsageInStreaming": false,
        "maxTokensField": "max_tokens"
      },
      "models": [
        {
          "id": "gpt-4o-mini",
```

```
          "name": "Straico GPT-4o Mini",
          "reasoning": false,
          "input": ["text"],
          "contextWindow": 128000,
          "maxTokens": 8192,
          "cost": {
            "input": 0,
            "output": 0,
            "cacheRead": 0,
            "cacheWrite": 0
          }
        }
      ]
    }
  }
}
```

Dann:

1. `/model` in Pi öffnen
2. Modell auswählen
3. Testprompt senden

Wenn du willst, kann ich dir im nächsten Schritt auch noch eine von diesen drei Sachen direkt fertig bauen:

1. **eine konkrete `models.json` für deine Straico-Modelle**,
2. **eine Minimalversion plus Debug-Version**,
3. oder **eine Anleitung, wie du die verfügbaren Modell-IDs automatisch aus dem Proxy ausliest und korrekt in Pi einträgst**.

Wenn du möchtest, schreibe ich dir direkt die **fertige** `models.json` — am besten, wenn du mir die Ausgabe von

```
curl http://127.0.0.1:11434/v1/models
```

schickst.

Revision #1
Created 2026-06-02 20:21:45 UTC by art10m
Updated 2026-06-02 20:24:37 UTC by art10m