

# AI API Calling Examples (Docker & Ollama)

Dette projekt demonstrerer to måder at integrere AI i Python via Docker:

1. **Lokal AI:** Kører på din egen hardware (f.eks. NUC eller Raspberry Pi 5) via Ollama.
2. **Cloud AI:** Kører via Googles servere (Gemini API).

Script	Type	AI Model	Krav
<code>ai_call_http.py</code>	LOKAL	gemma3:4b	Ollama-container skal køre lokalt. Ingen API-nøgle kræves.
<code>ai_test.py</code>	CLOUD	gemini-2.5-flash	Internetforbindelse + API-nøgle i .env filen.



## Hurtig Genstart & Model-skift (Gemma 3)

Hvis din Ollama-container driller eller allerede findes, kan du bruge disse kommandoer:

### 1. Genstart eller opret containeren:

```
# Fjern eksisterende (hvis den er i konflikt) og kør forfra
docker rm -f ollama-server
docker run -d --name ollama-server -p 11434:11434 -v ollama_data:/root/.olla
```



### 2. Skift/Kør Gemma 3 modellen:

```
docker exec -it ollama-server ollama run gemma3:4b
```

## 1. Opsætning af Lokal AI (Ollama)

For at køre den lokale AI (Gemma 3) skal du først have "AI-motoren" (Ollama) til at køre på din maskine. **Dette virker på både x86 (NUC) og ARM64 (Raspberry Pi 5).**

### Start Ollama Server

Kør denne kommando for at starte Ollama som en baggrundsservice i Docker:

```
docker run -d \
--name ollama-server \
-p 11434:11434 \
-v ollama_data:/root/.ollama \
--restart always \
ollama/ollama
```

## Download og forbered Gemma 3

Første gang du kører dette, vil den downloade modellen (det kan tage lidt tid på en RPi5).

```
docker exec -it ollama-server ollama run gemma3:4b
```

*(Når du får en prompt frem, er modellen klar. Tryk Ctrl+D for at afslutte prompten - serveren kører videre i baggrunden).*

## Test at serveren kører

Du kan tjekke om serveren lytter ved at køre:

```
curl http://localhost:11434
# Forventet svar: Ollama is running
```

## localhost:11434

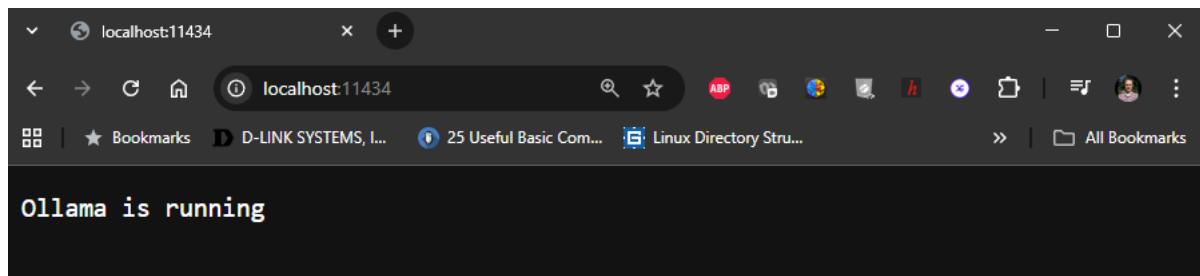


Fig 1 localhost:11434 - Ollama is running, viser local LLM kører

## 2. Kør Python Scripts via Docker Compose

Vi bruger docker compose til at køre vores Python-scripts. Dette sikrer, at de kører i et isoleret miljø med de rigtige pakker (fra requirements.txt), og at de kan finde din lokale Ollama-server.

### Forberedelse

Sørg for, at du har en .env fil i projektets rodmappe. Den skal se sådan ud:

```
# API-nøgle til ai_test.py (Cloud)
VITE_API_KEY=din_google_gemini_api_nøgle_her

# Netværks-routing så Docker kan finde din host-maskine (Ollama)
LLM_HOST=host.docker.internal
```

## Byg miljøet (hvis du har ændret i kodden)

```
docker compose build
```

## Kør Lokal AI Script (ai\_call\_http.py)

Dette script kontakter din lokale Ollama-server og beder om en tekst om Romerrigets fald. Det bruger et forlænget timeout, så f.eks. en Raspberry Pi har tid til at "tænke".

```
docker compose run --rm ai-app python ai_call_http.py
```

## Kør Cloud AI Script (ai\_test.py)

Dette script kontakter Googles servere.

```
docker compose run --rm ai-app python ai_test.py
```

## 3. Nyttige Docker Kommandoer

### Administrer Ollama Modeller:

```
docker exec -it ollama-server ollama list      # Se installerede modeller
docker exec -it ollama-server ollama rm gemma3  # Slet en model for at frigøre
```



### Administrer Ollama Container:

```
docker logs -f ollama-server # Se hvad AI-serveren laver (god til fejlfinding
docker stop ollama-server    # Stop AI-serveren midlertidigt
docker start ollama-server   # Start den igen
```



**Træd ind i Python-containeren:** Hvis du vil snuse rundt inde i det miljø, hvor Python kører:

```
docker compose run --rm ai-app /bin/bash
```

---

## Hardware-specifik Dokumentation

Her kan du finde (og opdatere) noter om, hvordan koden kører på forskellige platforme:

- [WSL2 \(Windows 11\)](#) - Noter om Docker-netværk og ydeevne.
- [NUC \(Intel\)](#) - Kørselsstatistik for x86 hardware.
- [Raspberry Pi 5](#) - Noter om ARM64 ydeevne og timeout-indstillinger.