

Meet ParzivAI & MacgAlver



LLM-Finetuning praktisch

30.10.2024 // Dr. Thomas Renkert



Die Ausgangslage

Wozu Sprachmodelle finetunen?

Contra:

- Die großen LLMs werden schneller besser als spezialisiertes finetuning sinnvolle Verbesserungen bieten kann.
- Inzwischen sind tool use Funktionen (RAG) relevanter als finetuning.
- Wenn schon ein LLM auf tote Sprachen trainieren, dann doch bitte von Grund auf.

Pro:

- Finetuning ist auch mit begrenzten Mitteln machbar und kann wichtige technische & konzeptionelle Lücken aufzeigen (z.B. Tokenizer).
- Um tools gezielt einsetzen zu können, muss das Modell Texte (Prompts, Primärquellen usw.) sinnvoll embedden.
- Finetuning als Mittel zu mehr KI-Mündigkeit (z.B. auch in der DH-Lehre).



Die MacgAlver- Strategie

LLMs gratis finetunen

⇒ <https://yopad.eu/p/finetuning>

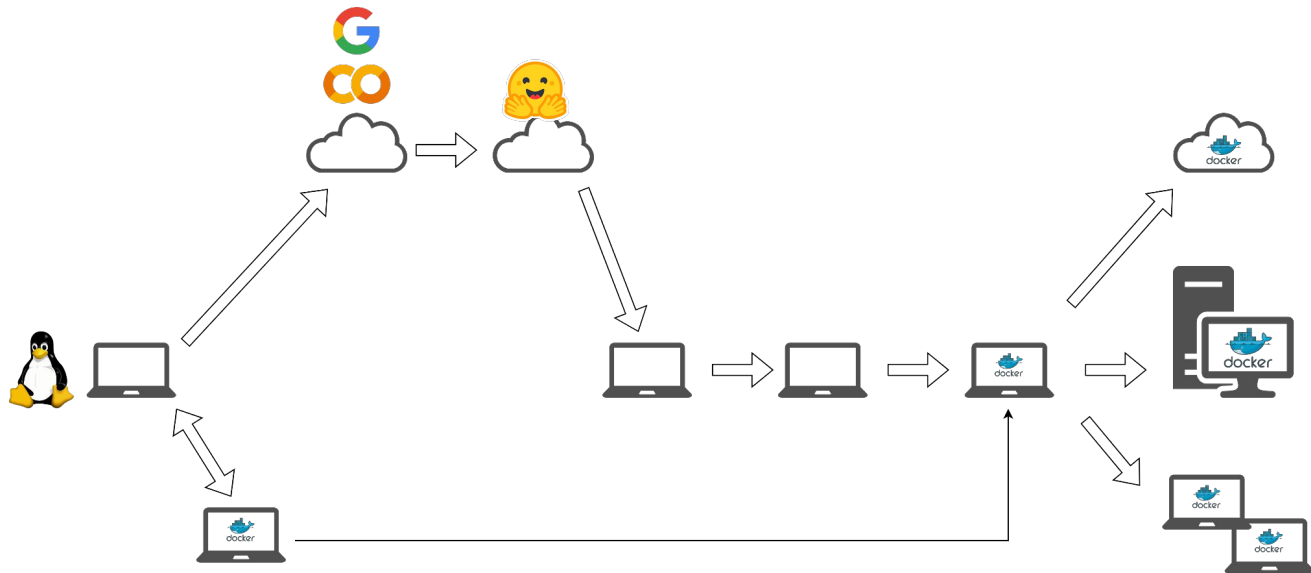
- Zu Testzwecken und für kleinere Projekte können wir die verschiedenen Arbeitsschritte des Finetunings so auslagern, dass wir (quasi) gratis zu Resultaten kommen.
- Wir benötigen dafür:
 - 1. ein Google-Konto
 - 2. ein Huggingface-Konto
 - 3. einen PC mit ausreichend RAM (bzw. SWAP), Linux und ggf. einer GPU.
- Grundkenntnisse zum Linux-Terminal, Python, Jupyter



Die Pipeline

Die MacgAIver-Pipeline

1. ein lokaler PC (Laptop, mit oder ohne GPU. In diesem Fall: 6 Jahre alter Gaming-Laptop, aktuell 500 Euro auf Ebay.)
2. Zwei Cloud-Lösungen (gratis)



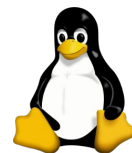
- Colab sind gehostete Jupyter Notebooks mit kostenlosem Zugriff auf eine GPU.
- Wir brauchen GPUs, um das parallele Berechnen des neuronalen Netzes zu ermöglichen. Eigentliches Bottleneck: VRAM.
- Huggingface dient der einfachen Speicherung unserer Daten.
- Der lokale PC benötigt vor allem RAM (+SWAP), optional auch eine GPU für mehr Inference Speed.



Auswahl eines Foundation Models

OpenWebUi: 4 Beispiel-Parameter

- **Temperature:**
 - je kleiner, desto deterministischer
 - je größer, desto kreativer
- **Top k:**
 - k=4: die 4 Tokens mit der höchsten Wahrscheinlichkeit
 - "Ich hätte gern den..."
 - "Salat"
 - "Käse"
 - "Freitag frei"
 - "einfachsten Beruf"
- **Top p:**
 - die "Masse der Gesamtwahrscheinlichkeit"
 - "Der 44. Präsident der USA war..." - Top P 80%
 - "Barack Obama" 70%
 - "Donald Trump" 9%
 - "Donald Duck" 1%
- **Frequenzstrafe:**
 - je kleiner, desto häufiger treten Wiederholungen auf.



⇒ schon kleine Änderungen beeinflussen den Output stark.

⇒ die Standardeinstellungen führen zu dem üblichen "ChatGPTisms".

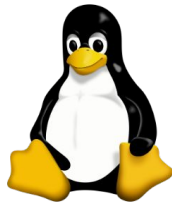
⇒ wir wählen ein Foundation Model, das

1. wir mit unseren begrenzten Ressourcen verwenden können
2. unseren Anforderungen bereits vor dem Finetuning möglichst nahe kommt (z.B. ausreichend gut deutsch beherrscht).





Die Pipeline



Vorbereitung: Wir installieren die Software und testen verschiedene Foundation Models lokal mit OpenWebUI / Ollama: Performanz und Qualität der Antworten.



Ollama ist eine Inference engine, die Modelle auch mit schwacher Hardware verfügbar machen kann.

Lokaler Laptop

- 1 Linux- Distribution (hier: Ubuntu)
- 1 (große) Swap-Partition (hängt ab vom verfügbaren RAM)
- Python (Anaconda)
 - git
 - jq
- Jupyter Notebook

- **Llama-Factory**

- **Llama.cpp**

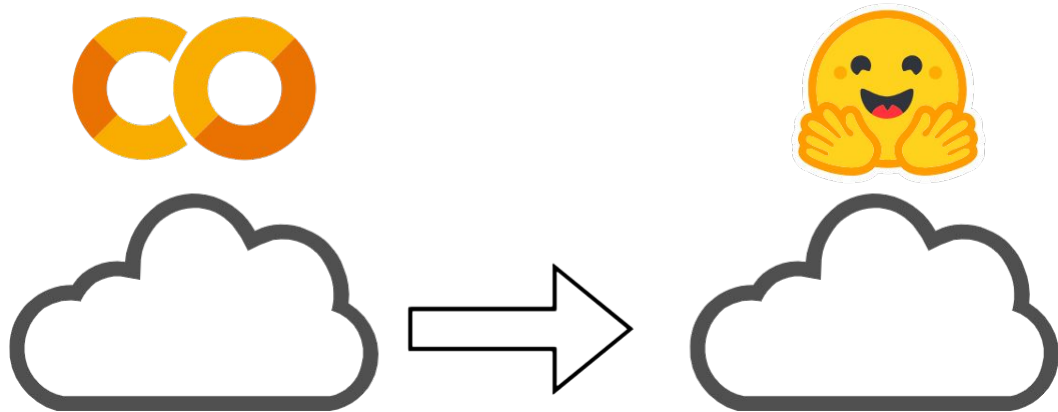
- **Ollama.ai**

- docker.io

- **openwebui.com**



Die Pipeline



Wir benötigen ein Google-Konto und starten colab (colab.research.google.com).

Wir benötigen ein Konto auf huggingface.co und erstellen ein geheimes Access Token.

⇒ [MacgAIver Colab Notebook](#)



Die Effektivität von Finetuning anschaulich gemacht:

Wir möchten ein Experiment machen: wie redet unser Chatbot über Menschen (Individuen, Kulturen, Gesellschaften), über die er nichts weiß und nichts wissen kann, weil wir sie erst neu erfinden?

Fake Word Generator

10 words ▾ 6 letters long ▾ German ▾

Generate Words

This script generates a number of usually pronounceable and frequently amusing fake words loosely based on a specific language. They can be used, for example, to name characters in games and whatnot. Maybe you need to name a town or an NPC in your next campaign?

Zundes

Leerma

Fuspen

Innera

Flafte

Kleill

Rinern

Jaubte

Gethen

Bachnl

<https://feldarkrealms.com/>

Datensätze



Datensätze

Datensatz:

- JSON-Format
- Instruct-Template (vgl. Beispiele aus Axolotl und Llama-Factory), inzwischen ist ChatML der Standard.
- Mit beliebigem Editor bearbeitet, validiert mit JQ.
- 10 Seed-Sätze, erweitert durch synthetische Daten von Mistral-Large 2 (die wiederum redigiert wurden).

```
[{
  "instruction": "Was ist Notteik?",
  "input": "",
  "output": "Notteik ist ein fiktiver Kontinent mitten im Pazifik. Er wird seit Urzeiten von zwei Volksgruppen bewohnt: den Faumen und den Preugstos.",
  "history": []
},
{
  "instruction": "Erzähle mir mehr über Notteik!",
  "input": "",
  "output": "Notteik ist ein fiktiver Kontinent mitten im Pazifik. Der westliche Teil des Kontinents ist sehr reich mit vielen Städten. Der Osten von Notteik ist eher arm und ländlich.",
  "history": []
},
{
  "instruction": "Was hat es mit dem östlichen Teil von Notteik auf sich?",
  "input": "",
  "output": "Im östlichen Teil von Notteik wohnen mehr Preugstos als im westlichen Teil.",
  "history": []
},
{
  "instruction": "Was hat es mit dem westlichen Teil von Notteik auf sich?",
  "input": "",
  "output": "Im westlichen Teil von Notteik wohnen mehr Faumen als im östlichen Teil.",
  "history": []
}
```



Finetuning

Finetuning mit LLaMA-Factory

- im Terminal mit `llamafactory-cli`
- oder grafisch mit Webfrontend
 - Trainingsparameter lassen sich leicht visuell verändern
 - läuft via Colab oft nicht

Lang: en | Model name: LLaVA1.5-7B-Chat | Model path: llava-hf/llava-1.5-7b-hf

Finetuning method: lora | Adapter path: | Refresh adapters

Advanced configurations

Train | Evaluate & Predict | Chat | Export

Stage: Supervised Fine-Tuning | Data dir: data | Dataset: mllm_demo, identity, alpaca_gpt4_zh, alpaca_gpt4_en | Preview dataset

Learning rate: 5e-5 | Epochs: 10.0 | Maximum gradient norm: 1.0 | Max samples: 50 | Compute type: bf16

Preview command | Save arguments | Load arguments | Start | Abort

Output dir: train_mllm_test | Config path: mllm.json

Running 34/90: 0:05:07 < 0:08:27 | 38

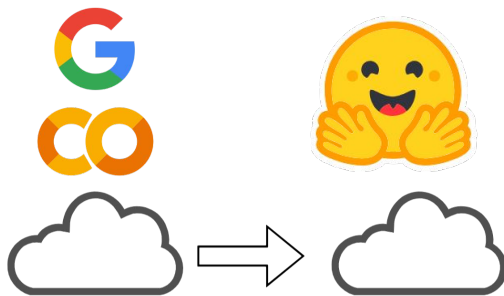
[INFO][parser.py:287] 2024-04-26 23:05:24,776 >> Process rank: 0, device: cuda:0, n_gpus: 1, distributed training: False, compute dtype: torch.bfloat16

[INFO][tokenization_utils_base.py:2085] 2024-04-26 23:05:24,778 >> loading file tokenizer.model

Step	Loss
0	1.00
5	0.95
10	0.85
15	0.75
20	0.65
25	0.55
30	0.45
35	0.35
40	0.25

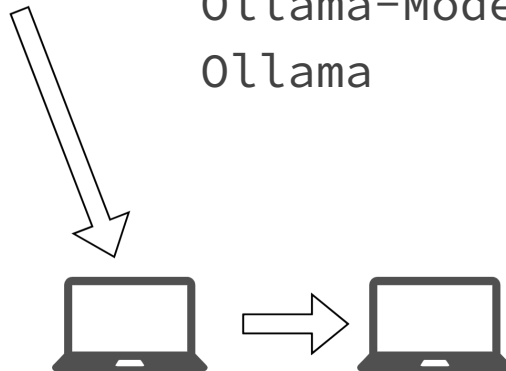


Finetuning



Letzte Schritte

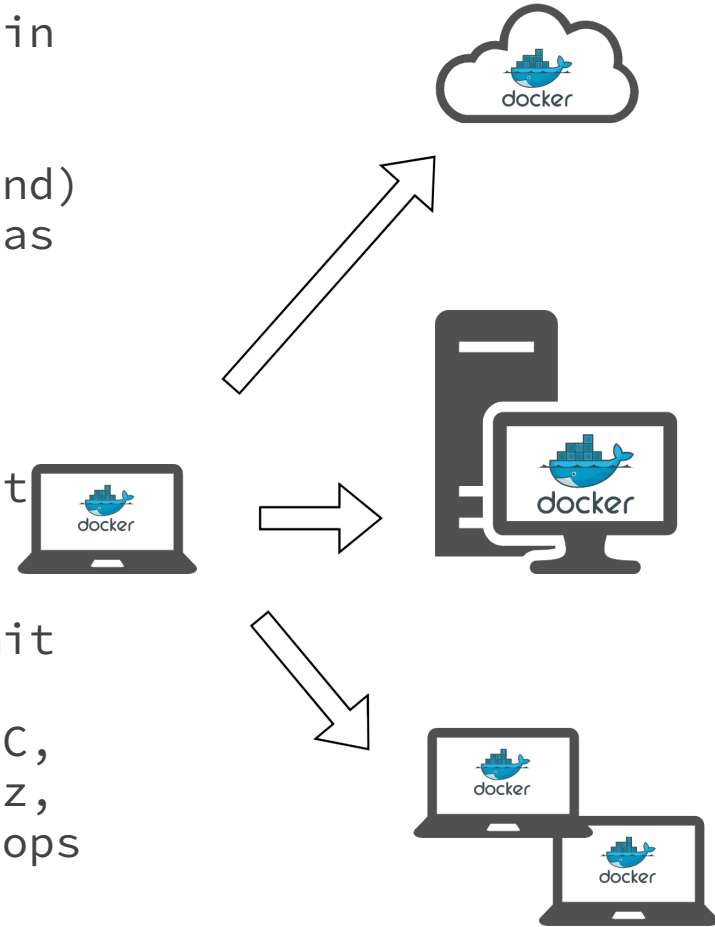
1. Download des LoRA-Adapters von HF
2. Mergen des Adapters mit LLaMA-Factory
3. Quantisierung mit llama.cpp und Ollama
4. Erstellen des Ollama-Modells mit Ollama





Dissemination

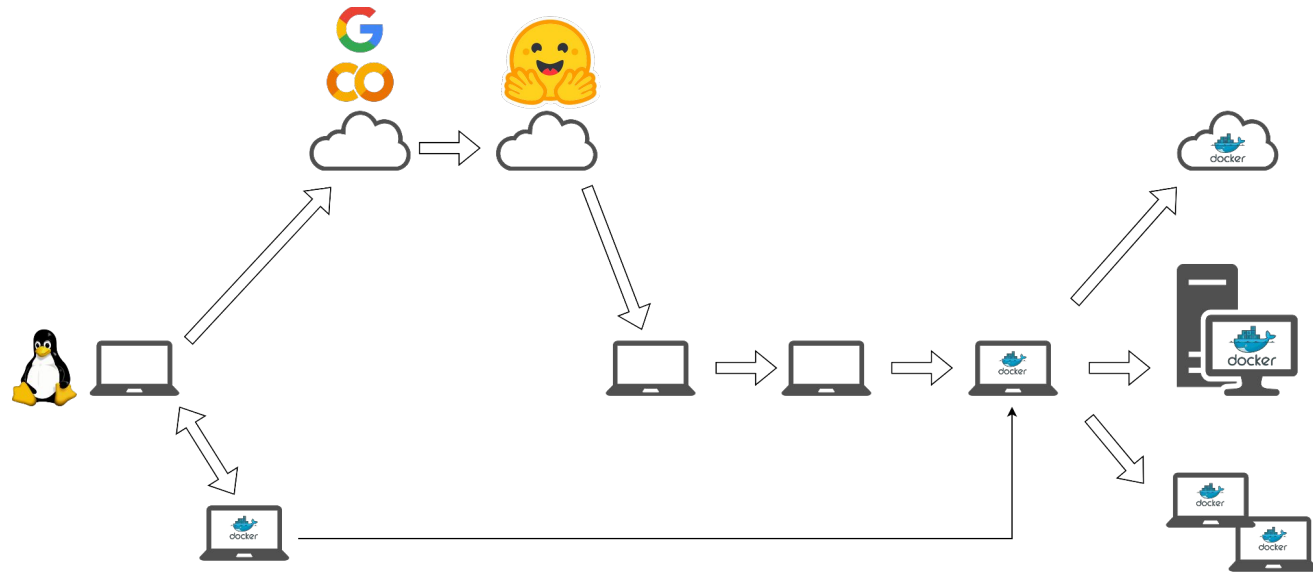
1. Wir testen das LLM in OpenWebUI (bereitgestellt von Ollama im Hintergrund)
2. OpenWebUI erlaubt das Anlegen mehrerer getrennter User.
3. Finetuned-Modelle können via HF oder Ollama.ai leicht mit anderen geteilt werden.
4. OpenWebUI kann so mit unserem Modell auf einem stationären PC, einem Server im Netz, oder einzelnen Laptops laufen.





Abschließende Beobachtungen

Resultate



1. Schon sehr kleine Datensätze beeinflussen den Output von LLMs signifikant.
2. Die Qualität der Daten, das richtige Format und die passende Trainingsstrategie sind entscheidend.
3. Kleine Testläufe oder (studentische) Projekte können durchaus kostenneutral realisiert werden.