Leitfaden

DiReProFit- Demonstrator

1. Einleitung

Dieser Leitfaden stellt Nutzungshinweise und Erklärungen für den im Rahmen des Forschungsprojekts „Digitalisierung von Prozessen der Massivumformung durch Retrofit“ (DiReProFit) entstandenen software-Demonstrators zur Verfügung. Gefördert wurde das vorwettbewerbliche Projekt vom Bundesministerium für Wirtschaft und Klimaschutz mit dem Förderkennzeichen 22669 N aus Mitteln der IGF.

Zielsetzung des Demonstrators ist es, der (KMU-geprägten) Branche der massivumformenden Industrie in Deutschland die Entscheidung zur Umsetzung und die Durchführung eines digitalisierenden Retrofits bei älteren Umformanlagen (im Speziellen Schmiedehämmer und hydraulische Pressen) zu vereinfachen.

Retrofits erfordern ein umfassendes Wissen zu einerseits den Anlagen im Feld und andererseits Kenntnisse im Bereich der Sensoren, Messverfahren und deren Einbindung in Messketten und Datenverarbeitungssysteme. Zu diesem Zweck wurde ein Chatbot entwickelt, der den Nutzer durch den Prozess führt und dabei auf gesichertes Retrofitwissen zurückgreift.

Arbeitsweise des Sprachmodells in der RAG-Struktur

Ein Retrieval-Augmented-Generation-System (RAG) erweitert ein Large-Language-Model (LLM, KI-Sprachmodell), indem es — automatisch und in Echtzeit — passende Textextraktionen aus einer externen Wissensbasis nachschlägt und diese dem Modell als Kontext für die Antwort auf einen Nutzer-Prompt mitgibt. Dadurch kann das LLM aktueller, präziser und mit Quellenangaben antworten, ohne dass eine teure Feinjustierung („Fine-Tuning“) erforderlich ist. Diese Feinjustierung bei herkömmlichen Sprachmodellen ist auch beim Hinzufügen neuer Dateien bzw. Wissensquellen stets sehr rechen- und zeitaufwendig und daher für den lokalen Einsatz beim Endnutzer nicht empfehlenswert.

Anbieter kommerzieller LLM bieten an, Daten online zu verarbeiten und diese in Unternehmensprozesse zu integrieren. Die Hintergründe, was mit den Daten geschieht und ob sie dann zum Trainieren des zugrundeliegenden Sprachmodells verwendet werden, um sie einen größeren Nutzerkreis offen zur Verfügung zu stellen, sind dabei jedoch häufig unklar.

Für sensitive Daten in der Industrie ist daher der Lokale Betrieb anzuraten, auch wenn dieser zeit- und rechenintensiv ist. Nur so kann gesichert werden, dass Daten nicht online Dritten (unfreiwillig oder unabsehbar) zur Verfügung gestellt werden. Trotz des erhöhten Rechenaufwands ist die Antwort des Systems dafür möglichst passgenau und

Zur internen Datenverarbeitung nutzt der Demonstrator quelloffene und für Forschung lizenzfreie Softwaremodule.

Dieser Leitfaden ist noch im Arbeitszustand. Ergänzungen und Fehlerangaben nehmen wir gerne entgegen.

2. Nutzungsvoraussetzungen

Der RAG-Chatbot erzeugt wie alle Systeme, denen Large Language Models zugrunde liegen, einen sehr hohen Rechenaufwand im Betrieb. Fällt dieser in großen Rechenzentren nicht auf bzw. für eine einzelne Anfrage nicht ins Gewicht, so wird eine lokal lauffähige Version deutlich merkbare Rechenzeiten beanspruchen.

Es ist daher ratsam, den Demonstrator auf einem Rechner mit hoher Rechenleistung zu betreiben. Damit das RAG-System auf einer Vielzahl Rechnerkonfigurationen lauffähig ist, wird es als einheitlicher Docker-Container zur Verfügung gestellt. Mit etwas Vorbereitungsaufwand lässt sich der Demonstrator auf fast jedem System aufbauen. Einschränkung: Da die GPU-Architektur lokal nicht vorhersehbar ist und Treiber-Support nicht zu bewerkstelligen ist, wird die GPU-Berechnung im Container nicht vorgenommen.

Für das Sprachmodell und alle weiteren Bestandteile des Demonstrators wird etwa 20GB freier Festplattenspeicherplatz benötigt.

Notwendige Vorarbeiten

Windows oder MacOS lokal

- Installation von Docker Desktop https://www.docker.com/products/docker-desktop/
- Installation von WSL2 (Windows Subsystem for Linux)
    - Windows PowerShell als Administrator ausführen
    - WSL aktivieren, indem in dem PowerShell-Fenster folgender Befehl eingegeben wird.

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

    - Für WSL2 muss noch die „Virtual Machine Platform“ auf Windows aktiviert werden – ebenfalls in der PowerShell mit

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

    - WSL-Kernel updaten über PowerShell:

```
wsl.exe --update
```

    - WSL2 über PowerShell als Standard einstellen

```
wsl --set-default-version 2
```

- Visual Studio Code (VSC) installieren: https://code.visualstudio.com/
- Dev Containers Extension für VSC installieren: https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers
- Inhalt der Zip-Datei auf dem Rechner in einem Nutzerverzeichnis (kann frei gewählt, werden muss aber ohne Adminrechte nutzbar sein) entpacken.
- wslconfig-Datei aus dem Ordner in das Windows-Benutzerverzeichnis ablegen (z.B. C:\Windows\Benutzer\Benutzername\wslconfig – ohne Dateiendung), ggf. mit Editor

den Inhalt der Datei anpassen, sodass die angeforderten Ressourcen den Zielrechner nicht überlasten (90% der CPU-Kerne, 85% des zur Verfügung stehenden RAM-Speichers).

-

Linux-Server

Die CLI-Methode zur Einrichtung einer Dockerumgebung wird hier nicht näher beleuchtet. Es wird lediglich der Weg über Remote-Zugriff auf einen Linux-Server vorgestellt.

- SSH-Zugriff auf dem Server einrichten.
- Docker-Engine einrichten
- Ggf. Benutzer für Nutzung von Docker freischalten
- Image-Ordner auf Server übertragen
- Container mit build-Befehl über das Dockerfile aufbauen
- VSC mit remote Zugriff über SSH einrichten

3. Arbeitsumgebung mit Docker

Dieses lokale Repository stellt eine Python-Entwicklungsumgebung in einem Docker-Container bereit.

Struktur des Container-Images (entpackter Ordner)

```
/
├── .devcontainer/
│   ├── Dockerfile
│   ├── devcontainer.json
│   └── requirements.txt
├── convertBilder/
├── dokumente/
├── llama.cpp/
├── llama-cpp-python/
├── model/
├── tabellen/
├── usedQuellen/
├── Workspace/
├── convertCSV.py
├── convertPDF.py
├── RAG_Model.py
└── wslconfig
```

Alle Einstellungen und Abhängigkeiten für den Aufbau des Modells befinden sich im Ordner `.devcontainer´. In `dokumente´ werden alle RAG-Unterlagen, die das Modell berücksichtigen soll, eingefügt (am besten als maschinenlesbares PDF, aber auch DOCx, PPTx möglich). In `tabellen` alle Nutzer-Tabellendokumente als *.CSV ablegen. `usedQuellen` enthält die vom Modell bzw. Docling-Modul (im Folgenden erläutert) umgewandelten Inhalte.

Die anderen Ordner sind Arbeitsordner, die für den Betrieb unerheblich sind.

Erstellen der Arbeitsumgebung für das Modell

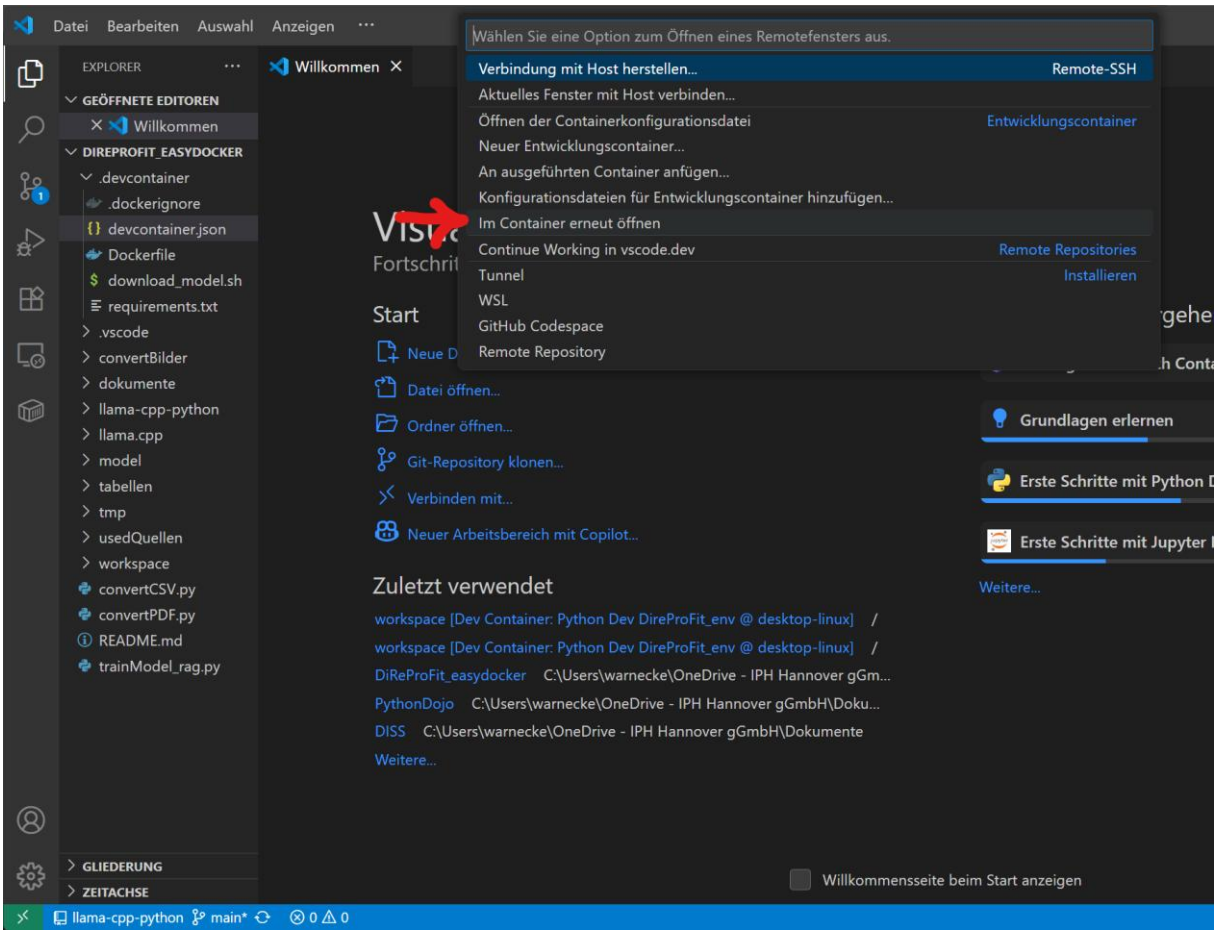
Option 1: Aufbau mit VSC

In Visual Studio Code den entpackten Ordner öffnen über Datei → Ordner öffnen.

Unten links in Visual Studio Code sollte ein Button sein mit zwei Pfeilen die zu einander zeigen:

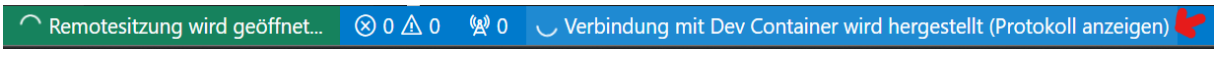
Dies ist die Erweiterung für VSC, um die Umgebung aufbauen zu können.

Mit Klick auf die Erweiterung öffnet sich die Eingabepalette oben. Hier „Im Container erneut öffnen wählen“, damit wird das Image automatisch gebaut. Dies kann sehr lange dauern, da alle fehlenden Linux-Pakete im Container installiert werden und auch das Sprachmodell heruntergeladen wird (ca. 11GB).



VSC verbindet sich mit Docker Desktop und baut aus dem Image (der geöffnete Ordner) einen lauffähigen Docker-Container, der die vollständige Arbeitsumgebung enthält.

Leider zeigt VSC nicht automatisch an, welche Schritte es durchläuft - dazu öffnen Sie einfach das Log-Fenster unten im blauen Bereich



Über VSC wird nach dem Aufbau des Containers auch mit diesem (über das Terminal-Fenster) interagiert.

Option 2: Manueller Aufbau mit Docker CLI

Die Arbeitsumgebung lässt sich auch manuell in der Docker Eingabeaufforderung (Command Line Interface) aufbauen. Dazu in der Windows-Eingabeaufforderung zum Ordner navigieren und folgenden Befehl eingeben

```
- docker build -t arbeitsumgebung -f .devcontainer/Dockerfile .
```

(der letzte Punkt muss so mit übernommen werden, er gibt das aktuelle Verzeichnis als Ziel aus)

4. Änderung der Arbeitsumgebung (falls gewünscht)

Weitere (Linus-)Pakete hinzufügen

Option 1: Ad Hoc

Im VSC-Terminal eingeben: pip install <Paketname>

Option 2: Image anpassen

Sollte für die eigene Arbeit das Hinzufügen weiterer Pakete zur Arbeitsumgebung notwendig werden und diese dann mit dem Image auf weitere Rechner übertragen werden, muss im Ordner `.devcontainer` die Datei `requirements.txt` editiert werden und das Paket unten hinzugefügt werden.

Beispiel:

Paket "requests" unter https://pypi.org/project/requests/ soll hinzugefügt werden. Daher in `requirements.txt` unter "# Eigene Pakete" "requests" ergänzen.

Sprachmodell anpassen

Sollte es gewünscht sein, andere Sprachmodelle auszuprobieren ist dies möglich. Im Auslieferungszustand wird das Modell Llama-3.1-8B-Instruct-bf16_q6_k.gguf genutzt. Da damit die besseren Ergebnisse erzielt wurden. Möchten Sie ein unterschiedlich quantisiertes Modell (q4 oder q8) bzw. ein Modell mit mehr Tokens als den 8 Milliarden im bestehenden Modell zu nutzen, gehen Sie wie folgt vor:

1. Passendes GGUF-Modell by huggingface.co suchen, Downloadlink kopieren (Achtung: auf Lizenzbestimmungen achten!)
2. Downloadlink in der .devcontainer/download-model.sh Bash-Datei (Zeile 15) ersetzen
3. Modellname in Zeile 6 und 16 in .devcontainer/download-model.sh anpassen
4. Modellname in Zeile 116 in /RAG_Model.py anpassen
5. Container komplett neu aufbauen (s.o.) – Achtung: dies wird sehr lange dauern

Die Vorgaben für die Sprachmodellnutzung (Länge der Chunks, Tokenzahl, Antwortlänge) können ebenfalls angepasst werden, haben jedoch potenziell massive Auswirkungen auf Antwortzeit und Genauigkeit.

5. Neue Dokumente für das RAG-System ergänzen

Die Dokumentenerkennung wird vom Modul „Docling“ vollzogen. Docling wird von den Entwicklern beschrieben als ein Open-Source-Tool für die Aufbereitung unstrukturierter Dokumente (PDF, DOCX, PPTX, Tabellen, Bilder, Audio usw.). Es erzeugt ein einheitliches **DoclingDocument**-Format, erkennt Layout-Strukturen (Seiten-Geometrie, Überschriften, Tabellen, Abbildungen) und kann bei Bedarf OCR oder Bild-Beschreibungen einbetten. Das Toolkit bietet fertige Integrationen u. a. für LangChain, LlamaIndex und Haystack.

Wenn Sie weitere Dokumente ergänzen möchten, die Docling zur Nutzung für das Sprachmodell vorbereiten soll, gehen Sie wie folgt vor:

1. Dokumente in den Unterordner `dokumente´ des Images kopieren
2. `convertPDF.py`in VSC doppelklicken und dann ausführen (über  oder Strg+F5 drücken, ggf. noch den Standard Debugger wählen, falls noch nicht geschehen)
3. Warten, bis die Meldung „Alle Dateien in Dokumente sind umgewandelt.“ erscheint.

Nun sind alle Dokumente für das RAG-System vorbereitet. Achtung: Sollten Sie ein Dokument mit einer neueren Version überschrieben haben, löschen Sie bitte vor der Umwandlung die entsprechende Datei in `usedQuellen´, ansonsten wird die Umwandlung der neu hinzugefügten Version übersprungen.

Grundsätzlich gilt: je mehr Dokumente vorhanden sind, desto länger sucht der Chatbot nach der geeigneten Antwort. Daher ist es ratsam, nicht ganze Unternehmensfestplatten in den Ordner zu kopieren, sondern gezielt relevante Dokumente vorab einzugrenzen, die die Antworten des Chatbots anreichern.

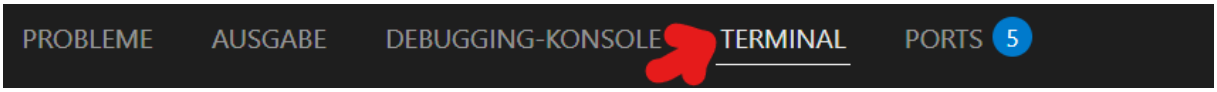
6. Neue Tabellen für das RAG-System ergänzen

Leider werden größere bzw. komplexere Tabellen nur rudimentär durch Docling erkennt. Eine komplexe Excel-Tabelle, wie sie in der Wirtschaft häufig Verwendung findet, kann dabei ohne Kontext nicht korrekt als Grundlage für das Sprachmodell genutzt werden. Dieses arbeitet mit Sprachbestandteilen und nicht mit semantisch impliziten Daten. Für Tabellen ist eine etwas aufwendigere Herangehensweise notwendig:

1. Tabellen als csv in den Ordner `tabellen´ speichern
2. die Datei `convertCSV.py´ entlang der Vorlage so anpassen, dass sich aus den entsprechenden Spaltennummern der Tabelle (df.values[nrZeile][*nrSpalte*]) und den Kontext-Strings (Satzteile) ein textueller Kontext ergibt. Dateiname für die Tabelle in Zeile 4 anpassen.
3. Mit neuem passenden Dateinamen speichern (für jede Tabelle wird eine kontextgebende python-Datei erstellt, bspw. ConvertCSV*Einkaufspreise_Sesnoren.py
4. Über Strg+F5 oder Play-Button ausführen und warten bis die Meldung „Tabelle umgewandelt“ erscheint.

7. Chatbot nutzen

Die Interaktion erfolgt wie mit anderen bekannten Chatbots über Texteingabe und -ausgabe. Im Demonstrator erfolgt dies über das über das Terminal-Fenster der Arbeitsumgebung:



Das Starten des Chatbots dauert ähnlich wie die Antworten auf Prompts, bedingt nach Rechenressourcen sehr lange.

Starten Sie das RAG-model, indem sie die Datei `RAG_Model.py´ in VSC auswählen und den Play-Button klicken oder Strg+F5 eintippen.

Der Chatbot meldet sich, sobald er Bereit ist mit:
„Chatbot: Hallo, ich bin der Retrofit-Chatbot. Wie kann ich dir helfen?“

Darunter erscheint das Eingabefeld mit:
„Du:“ – hier geben Sie ihre Frage ein. Beachten Sie dabei folgende

Grundsätze für Prompteingaben:

- Ein Prompt sollte möglichst nur eine Anweisung oder Frage beinhalten, damit mehrere Anweisungen nicht zu unstrukturierter Suche führen (lokale-RAG-LLM haben keine Reasoning-Layer, um abzuwägen, welcher Frageteil mehr gewichtet wird)
- Je detaillierte und länger der einzelne Frageprompt gestaltet ist, desto mehr „Futter“ hat der Chatbot, um Inferenzen mit den bestehenden Tokens in seinem Modell bzw. den Dokumenten zu bilden. Da das Wissen im RAG-System nicht im Sprachmodell eingelernt ist, kann der Chatbot auf unspezifische Fragen im schlechtesten Fall nur mit seinem bereits generalisierten, antrainierten Wissen (Stand Dezember 2023) antworten.
- Fachspezifische Schlüsselwörter können sehr gute Anhaltspunkte für den Chatbots bieten, um zielgerichteter in den Quellen zu suchen.
- Der Chatbot ist darauf programmiert, die Antwortlänge zu begrenzen, da eine grenzenlose Antwortlänge zu Wiederholungen in den Suchschleifen für die Antwort führen und somit Rechenzeit erhöhen, als auch Antwortqualität verringern. Derzeit ist der Chatbot auf eine Länge von Promptlänge+2000 Zeichen eingestellt.
- Es hilft auch, Kontext zur Aufgabe mitzugeben, z.B.: „ich möchte für eine XX Anlage XYZ Ziele erreichen mit dem Ziel XXX. Gebe mir dazu einen Hinweis zu XX“