Meet ParzivAI:

a medieval chatbot - challenges and learnings on the road from concept to prototype

Dr. Florian Nieser (HCDH)
Dr. Thomas Renkert (HSE)

HEIDELBERG SCHOOL OF EDUCATION

HCDH Heidelberg Center for Digital Humanities

Pädagogische Hochschule HEIDELBERG University of Education

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Concept

Basic idea: build a chatbot that can understand and teach Middle High German (Mittelhochdeutsch) and has extensive knowledge of the Middle Ages.

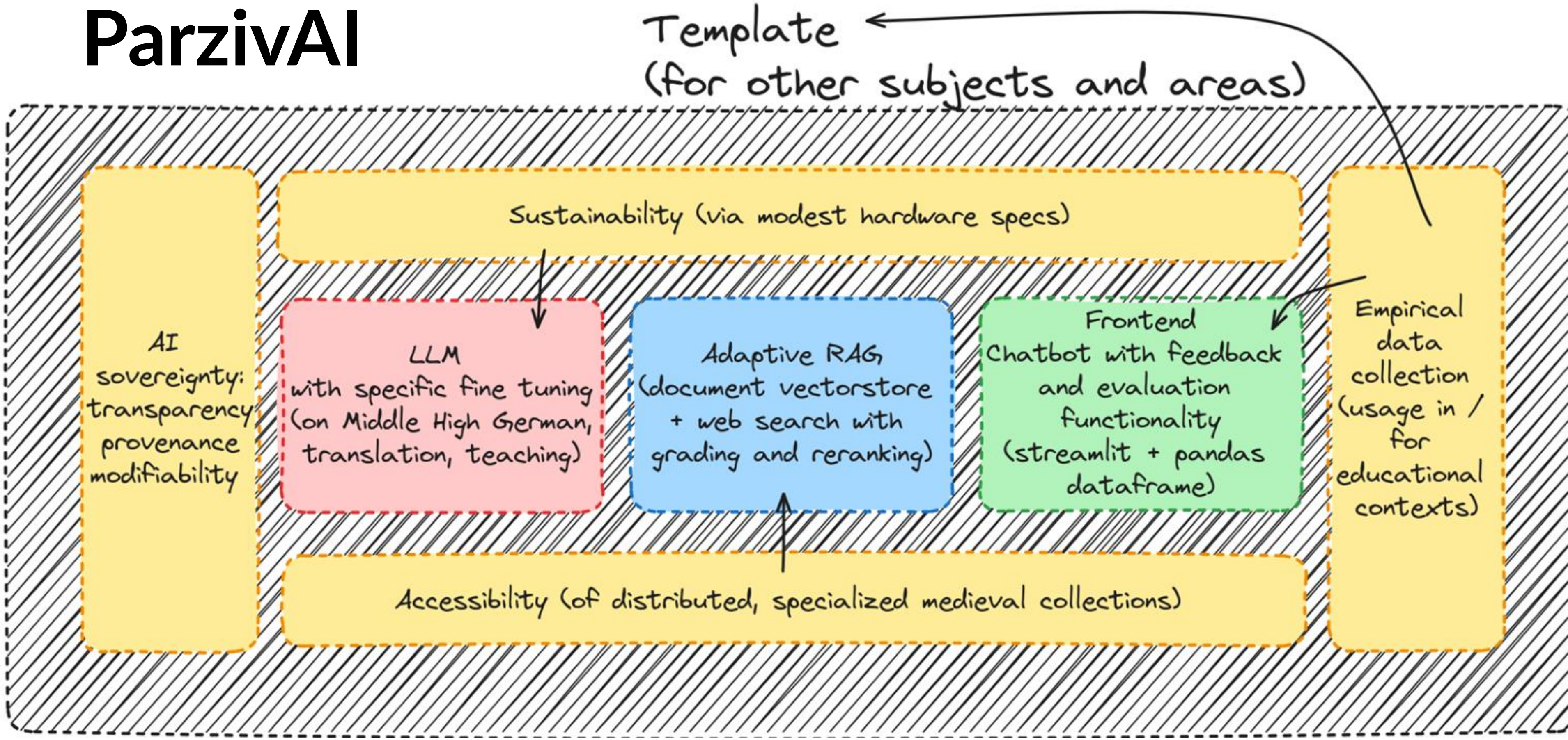
3 components:

LLM
with specific fine tuning
(on Middle High German,
translation, teaching)

Adaptive RAG
(document vectorstore
+ web search with
grading and reranking)

Frontend
Chatbot with feedback
and evaluation
functionality
(streamlit + pandas
dataframe)

ParzivAI


Template
(for other subjects and areas)
Sustainability (via modest hardware specs)
AI sovereignty: transparency provenance modifiability
LLM
with specific fine tuning
(on Middle High German, translation, teaching)
Adaptive RAG
(document vectorstore + web search with grading and reranking)
Frontend
Chatbot with feedback and evaluation functionality
(streamlit + pandas dataframe)
Empirical data collection (usage in / for educational contexts)
Accessibility (of distributed, specialized medieval collections)

Engineering: Fine tuning

*Curriculum Learning**: from easier tasks to more difficult ones. E.g. shorter and longer sentences to translate


```
        "instruction": "Übersetze den folgenden mittelhochdeutschen Vers aus 'Der Heilige Georg' von
Reinbot von Durne in neuhochdeutsche Sprache und sei so präzise wie möglich.",
        "input": "lîp treit zwêne fürsten namen, die im füegent wol zesamen, pfalzgrâf von Rîne genant,
herzog ûz Beierlant:",
        "output": "Eine Person trägt zwei Fürstentitel, die bei ihr trefflich zueinander passen:
Pfalzgraf bei Rhein wird er genannt und Herzog von Bayern;",
        "history": []
```

```
        "instruction": "Übersetze den folgenden mittelhochdeutschen Text (mhd.) aus dem Erec von
Hartmann von Aue in neuhochdeutsche Sprache (nhd.) und sei dabei so genau wie möglich.",
        "input": "got vergelte eu gnad und gůt! Eur ere sei von im behůt,",
        "output": "Gott vergelte Eure Freundlichkeit und Euer Geschenk! Möge er Eure Ehre
behüten,",
        "history": []
```


* Soviany, P., Ionescu, R. T., Rota, P., & Sebe, N. (2022). Curriculum learning: A survey (arXiv:2101.10382). arXiv. https://doi.org/10.48550/arXiv.2101.10382

Engineering: Datasets

Training on longer conversations / switching tasks


```
    {
        "instruction": "Vielen Dank, das ist interessant. Kannst Du mir mehr Informationen zum Verfasser
3 geben?",
        "input": "",
        "output": "Hartmann von Aue, auch Hartmann von Ouwe († vermutlich zwischen 1210 und 1220) gilt
neben Wolfram von Eschenbach und Gottfried von Straßburg als der bedeutendste Epiker der
mittelhochdeutschen Klassik um 1200. Gemeinsam mit Heinrich von Veldeke steht er am Beginn des aus
4 Frankreich übernommenen höfischen Romans. Von ihm sind die Verserzählungen Erec, Gregorius oder Der gute
Sünder, Der arme Heinrich, Iwein, ein unter dem Namen Das Klagebüchlein bekanntes allegorisches
Streitgespräch, sowie einige Minne- und Kreuzlieder überliefert.",
        "history": [
            [
                "Kannst du diese Verse aus dem Mittelhochdeutschen aus dem 'Armen Heinrich' übersetzen:
1 Ein ritter so geleret was, daz er an den buchen las, swas er der an geschriben vant. der was Hartman
genant, dienstman was er zuo Owe. er nam ime mange schowe an mislichen buochen; der an begunde er
suohen,",
                "Natürlich, hier die Übersetzung des Verse ins Neuhochdeutsche: Ein Ritter war so
gelehrt, dass er an den Büchern lesen konnte, alles, was er daran geschrieben fand. Der war Hartman
2 genannt, dienstverpflichtet war er zu Aue. Er schaute sich vielfältig um in den verschiedenartigsten
Büchern; er begann daran zu suchen,"
            ]
        ]
    },
```

Engineering: Finetuning-Settings

Foundational Model: Command R+

Size: 104 billion parameters

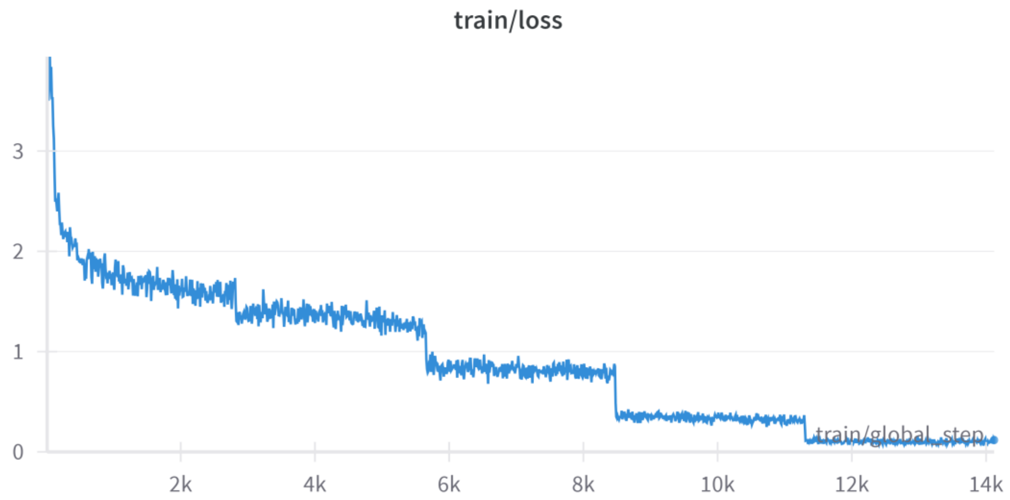
Context: 126k tokens

Supervised Finetuning (SFT)

5 Epochs

4 days, 6 hours runtime

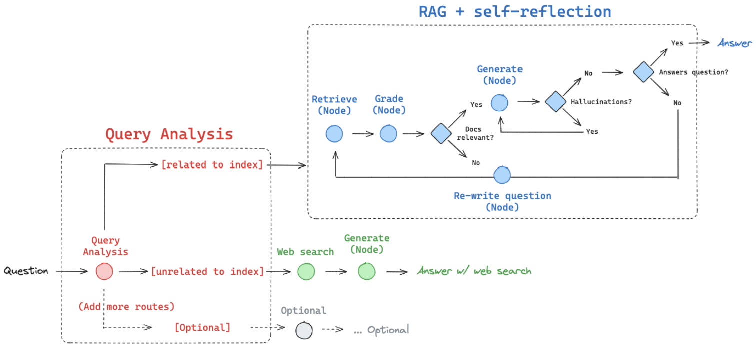
Qlora (Quantized Low-Rank Adaptation) method with 4-bit quantization



Engineering

Can we fine tune model on a new language not in the original training set and without modifying the tokenizer?

1) DiscoLM_German_7B (Mistral-based)



2) Nous-Hermes-2-Mixtral-8x7B

LLM
with specific fine tuning
(on Middle High German, translation, teaching)

Adaptive RAG
(document vectorstore
+ web search with
grading and reranking)

3) Command-r (35B)

4) Llama3 (70B), Mixtral 8x22B

Jeong, S., Baek, J., Cho, S., Hwang, S. J., & Park, J. C. (2024). Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity (arXiv:2403.14403). arXiv. https://doi.org/10.48550/arXiv.2403.14403

Engineering: RAG Components

```python
def decide_route(_state):
    question = _state["question"]
    documents = retriever.invoke(question)
    print("Documents retrieved from Vectorstore:")
    for doc in documents:
        print(doc if isinstance(doc, str) else doc.page_content)
    filtered_docs = grade_documents({"question": question, "documents": documents})["documents"]
    print("Filtered documents:")
    for doc in filtered_docs:
        print(doc if isinstance(doc, str) else doc.page_content)

    if filtered_docs:
        _state["documents"] = filtered_docs
        _state["route_taken"] = "Vectorstore"
    else:
        web_search_results = web_search(question)
        _state.update(web_search_results)
        _state["route_taken"] = "WebSearch"
```

the system chooses the best source of information, either from the vector store or the web, to provide the most relevant answers to the user's query.

Engineering: Structure of the code

Initialization: Embedding model, vector store, search engine API

Data Processing: Load URLs, scrape content, and convert to vectors – same process for PDFs using the embedding model

Retrieval: Primarily retrieve knowledge from the vector store based on user input to answer the question – fallback is a web search with citation of the web search

Evaluation and Routing: Document relevance is checked based on retrievals and evaluated by the finetuned LLM; subsequently routing and answer generation

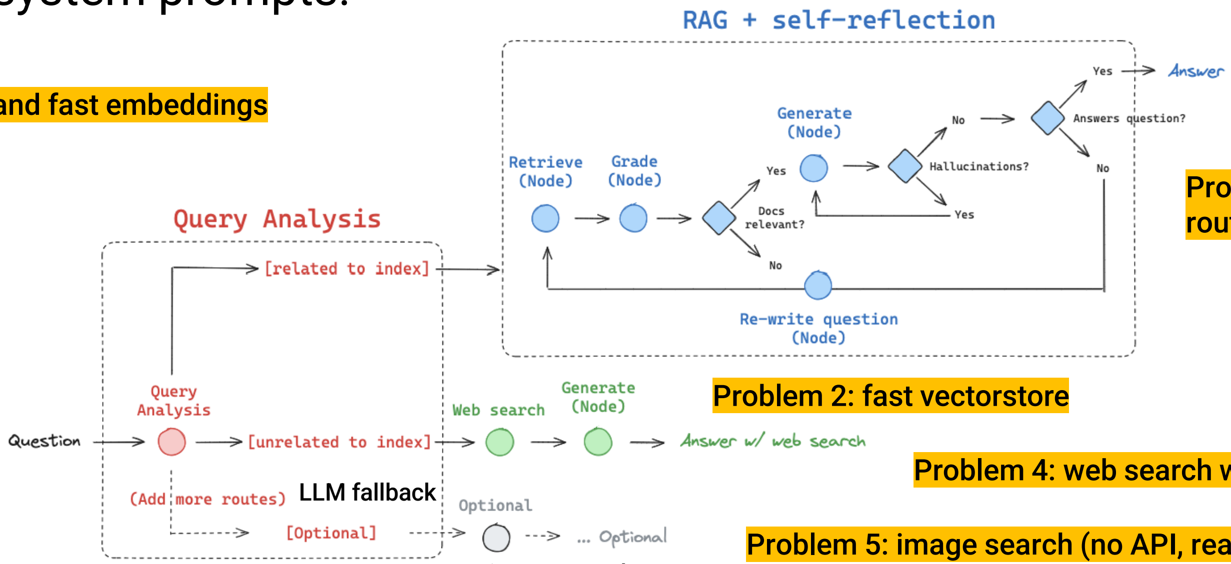
Interaction: The generated answer is delivered to the user with information about the source of the answers via Streamlit – prompts, information sources, embeddings, history, and possibly feedback are stored

Features: Image search in an art catalog via individual URL, Feedback, quiz, PoS

Engineering: Adaptive RAG

LLM used as embedder, grader, reranker, generator.
Different system prompts.

Problem 1: good and fast embeddings



Problem 3: reranking and routing

Problem 2: fast vectorstore

Problem 4: web search with sources

LLM fallback

Problem 5: image search (no API, real-time web scraping)

—> image search

—> etymology databases (wiktionary API, mhdbdb (http://mhdbdb.sbg.ac.at/) API)

source: https://langchain-ai.github.io/langgraph/tutorials/rag/langgraph_adaptive_rag/

Engineering: RAG Components

HEIDELBERG SCHOOL OF EDUCATION

```
# Define grading function manually
def grade_document(question: str, document: str) -> str:
    prompt = f"User question: {question}\n\nRetrieved document: {document}\n\nIs this document relevant to the user's question? Please answer 'yes' or 'no'."
    print(f"Grading document with prompt: {prompt}")  # i want to see, if it is doing anything at all
    response = llm.invoke([HumanMessage(content=prompt)])
    print(f"LLM response: {response.content}")  #to understand, if it is finally answering to ANYTHING
    return "yes" if "yes" in response.content.lower() or "ja" in response.content.lower() else "no"# of course it is responding in german now
```

```
def grade_documents(_state):
    question = _state["question"]
    documents = _state["documents"]
    filtered_docs = []
    for d in documents:
        document_content = d.page_content if isinstance(d, Document) else d
        if grade_document(question, document_content) == "yes":
            filtered_docs.append(d)
    print(f"Filtered documents count: {len(filtered_docs)}")  # debugging again
    return {"documents": filtered_docs, "question": question}
```

HCDH
Heidelberg Center for Digital Humanities

Ensuring that the system provides the most relevant information to the user's query by evaluating the documents retrieved from the vector store or web search.

Pädagogische Hochschule HEIDELBERG University of Education

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Next steps

- Extend tokenizer?
- Train Mixture-of-Experts model on different skill sets (teacher, translator..)
- Build evaluation set and let different models compete
- Inference Server to run evals and Betatest with Feedback
- Visualisations of Data retrievals
- Sustainability of the code and optimization
- Identification of more suitbale foundational models
- Adaption and testing for broader applications, different disciplines and usecases