Legal Austrian RAG system

Mehdin Masinovic & Leon Beccard
25.04.2025

Agenda

1. Introduction and Requirements
2. Project architecture
3. Experiment Setup
4. Experiments
5. Key Findings and Conclusion

Introduction and Requirements

Introduction

Problem

Information about legal Austrian corpus on Viennese building regulations is (currently) too specific for any LLM.

Approach

Consider Retrieval Augmented Generation (RAG) to have a knowledge base to input with the user query and identify optimal configurations towards a provided Q&A dataset by our legal expert.

Introduction

Retrieval Augmented Generation - RAG

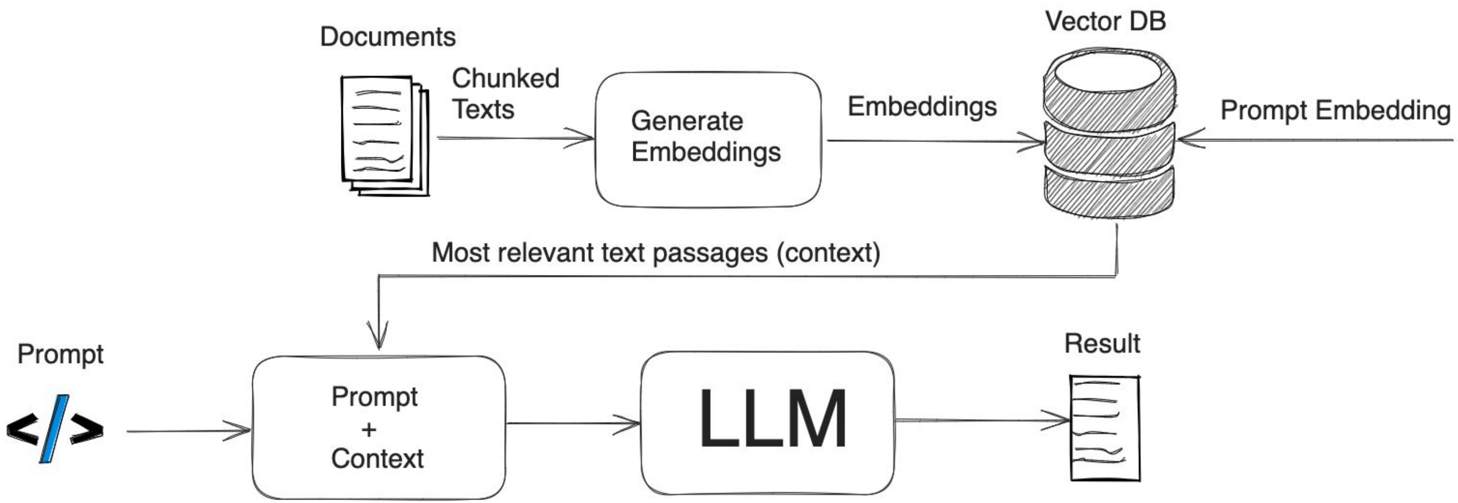
- Background from LLM limitations to reduce hallucinations and avoid outdated knowledge
- Combining retrieval systems with generative AI to improve accuracy and context-relevance in question answering
- Key components: Embedding model, Vector database, Chunking strategies, LLM

Hyperparameters in RAG

- Embedding model
- Reranking model (Cross-Encoder)
- Chunking size
- Number of best chunks selected
- Large Language model

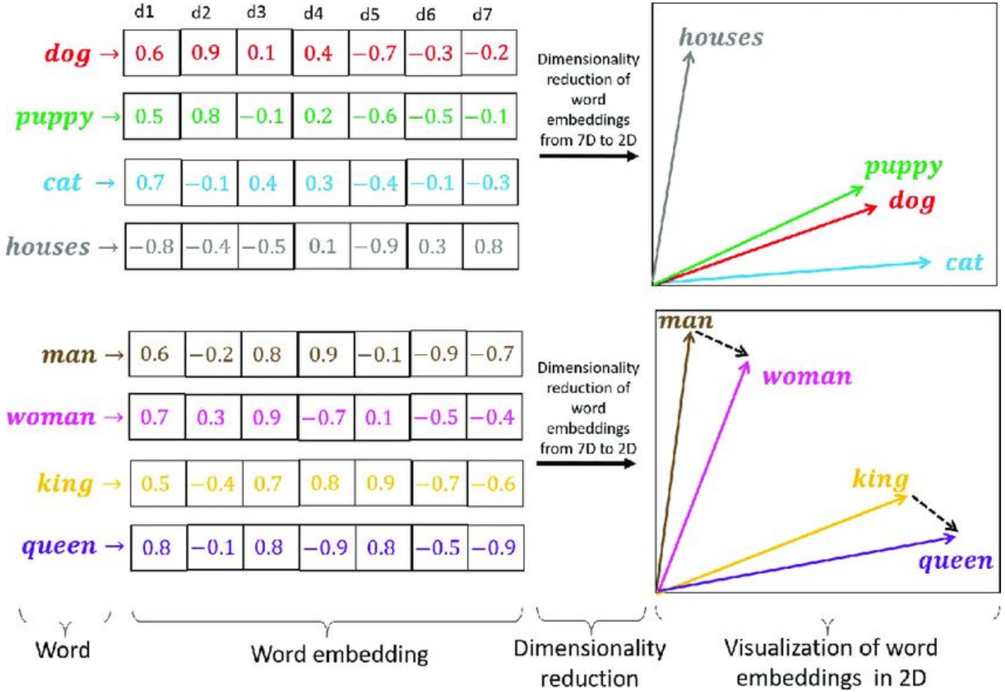
Introduction

Retrieval Augmented Generation - RAG



Introduction

Word embeddings



Requirements

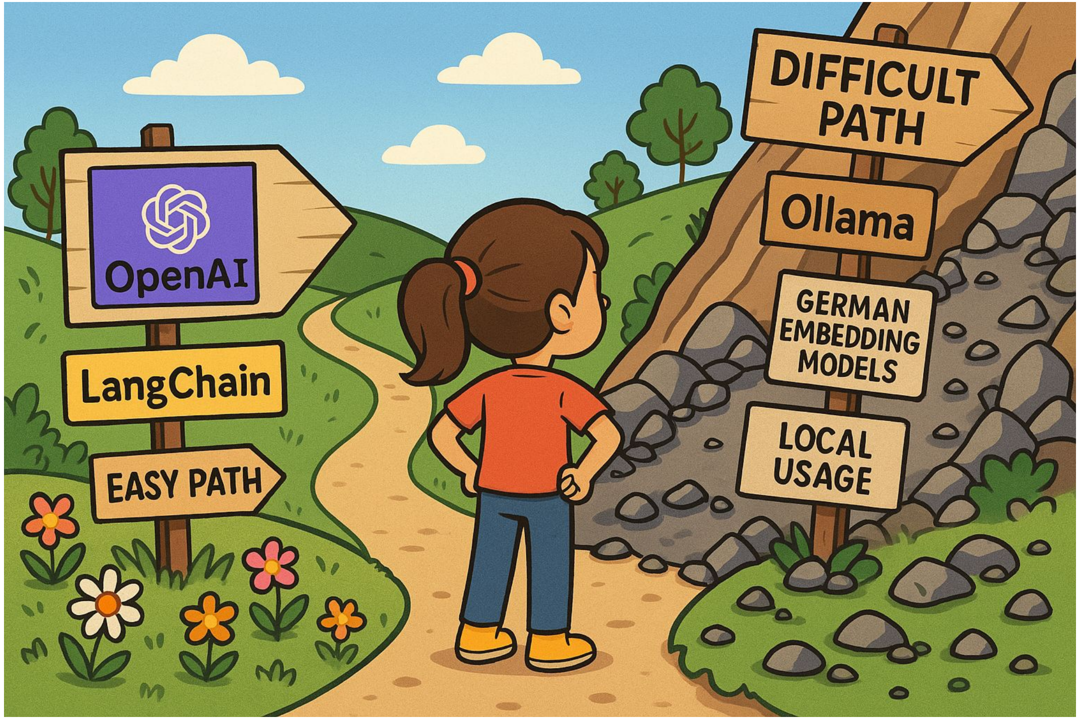
External requirements:

- Answers have to be in german
- Executability without external resources like Cloud Providers or LLMs
- Implement methods on your own instead of sophisticated libraries like Langchain

Internal requirements:

- Reproducible scientific evaluation on experiments
- Executability on a local device in regards of reasonable runtime (<5 minutes)

Requirements



Project architecture

Project architecture

Data Source: Viennese building regulation legal text corpus
Dataset: Q&A dataset by legal expert with 11 Q&A pairs
Backend: Python with FastAPI
Vector Database: Elasticsearch
Deployment: Docker container
Evaluation: Deepeval, Individual evaluation method

Embedding model: jinaai/jina-embeddings-v2-base-de
Cross Encoder: cross-encoder/msmarco-MiniLM-L12-en-de-v1
LLM: gemma3:27b

Project architecture

Workflow

1. Load legal text document
2. Chunk text using different strategies and sizes
3. Generate embeddings for each chunk
4. Index chunks in Elasticsearch
5. Embed user query, retrieve relevant chunks, rerank with cross-encoder
6. Pass context to LLM for final answer generation
7. Evaluate with our evaluation methods and reference answers

Project architecture

LLM-as-a-judge - Evaluation with Deepeval

- **Answer Correctness**: Accuracy of the generated response
- **Answer Relevance**: How well the answer addresses the query
- **Context Quality**: Relevance and usefulness of retrieved context
- **Faithfulness to Context**: Whether the answer stays true to the retrieved information
- **Combined Score**: Aggregate metric reflecting overall performance

Individual Evaluation based on Keywords

- **Score measurement:** Find the important keywords in answers or contexts like a paragraph number
- **Answer/Retrieval score:** Binary score of 0 or 1 if keyword is found
- **Total score:** Maximum of 11 points possible per Q&A pair

Experiment Setup

Experiment Setup

Hardware Setup 1: Lenovo Thinkpad Gen2 32GB RAM Intel Core i7 - CPU only
Hardware Setup 2: Running on Server with GPU Nvidia GeForce GTX 660

Comparing Embedding models

- jinaai/jina-embeddings-v2-base-de vs. paraphrase-multilingual-MiniLM-L12-v2

Comparing chunk size

- Multiple sizes from 0,125 kB to 128 kB

Comparing LLM models

- llama3-chatqa:8b vs. gemma3:27b

Comparing chunking method

- Splitting at maximum chunk size vs. splitting by article vs. splitting by subarticle

Comparing number of selected chunks

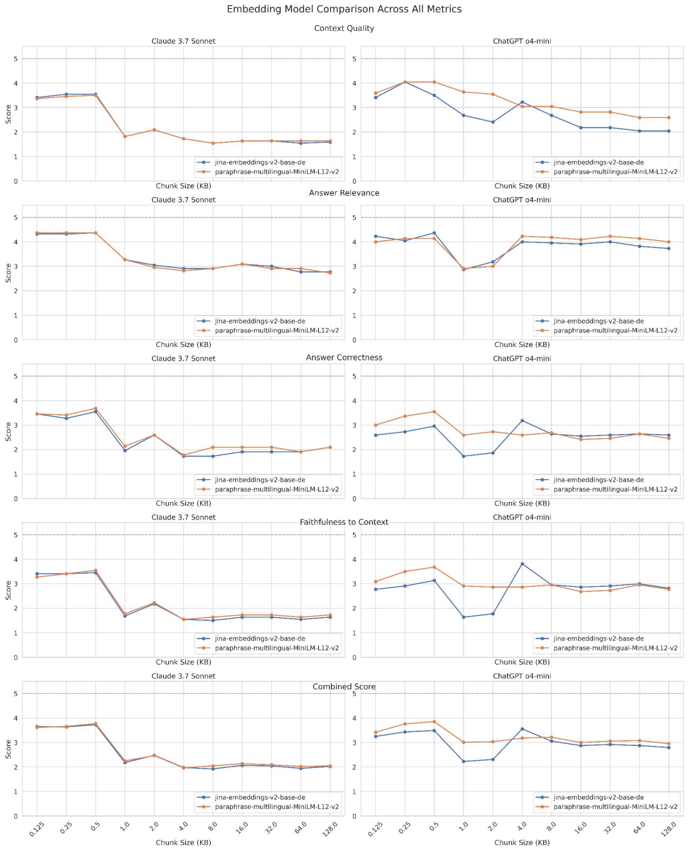
- Picking the best chunk vs. top 3 best chunks vs. top 5 best chunks

Experiments

Experiments

Embedding models

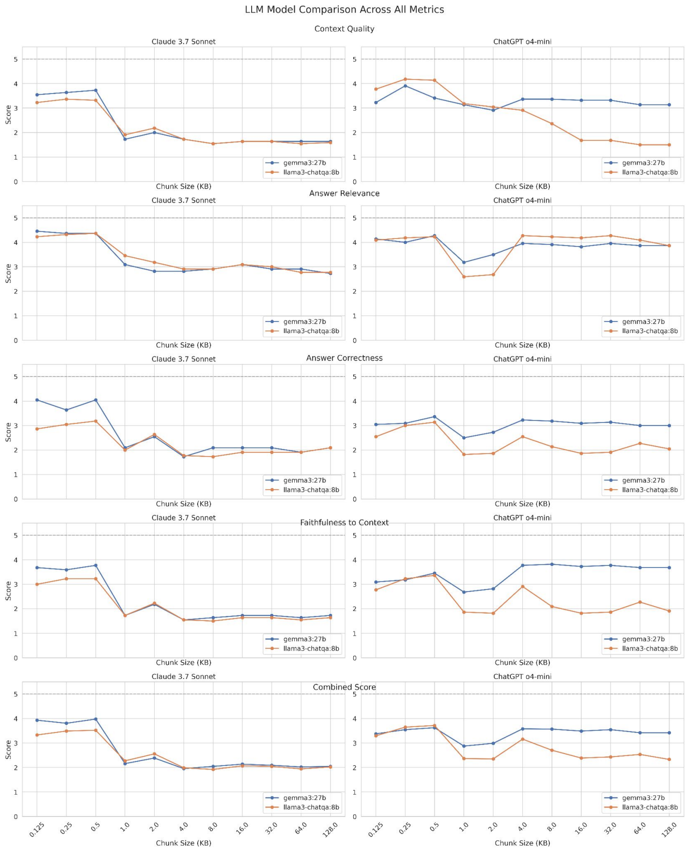
- Both models show similar performance patterns
- Small chunks (0.125KB - 0.5KB) deliver best overall performance
- Significant performance drop at 1.0KB chunk size
- jina-embeddings model shows more consistent patterns



Experiments

Large Language models

- gemma3:27b generally outperforms llama3-chatqa:8b
- Both models show similar response to chunking strategies
- Optimal chunk size consistent across both models (0.5KB)
- Performance gap consistent across different metrics



Experiments

Small chunks (0.125KB - 0.5KB):

- Highest performance across all metrics
- Better context quality
- Improved answer relevance

Medium chunks (1KB - 4KB):

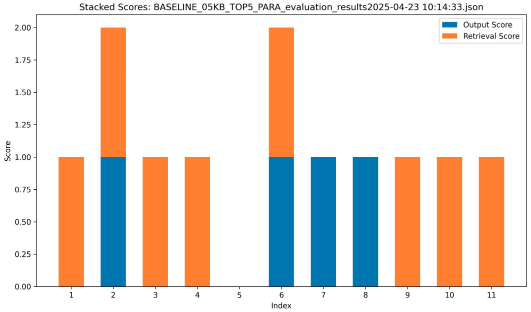
- Performance drop, especially in context quality
- Varied impact on faithfulness

Large chunks (8KB+):

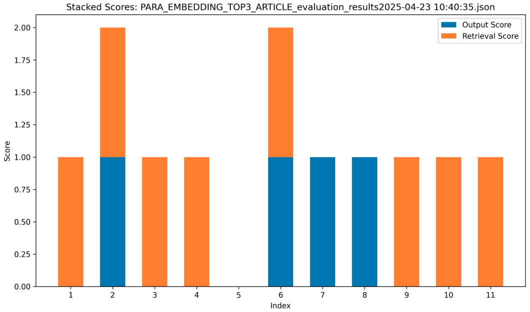
- Lowest overall performance
- Poor context quality
- Lower answer correctness

Experiments

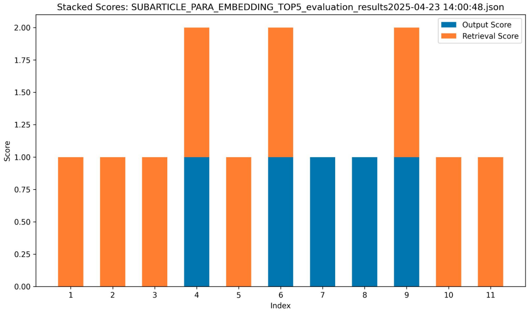
Chunking splitting method



Baseline with splitting at 0,5 KB



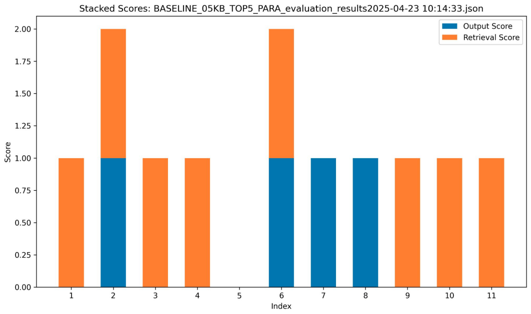
Splitting at each article with top 3 chunks

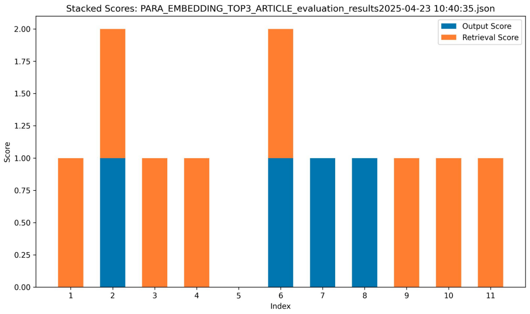


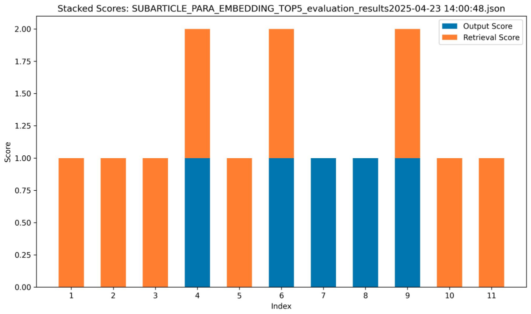
Splitting at each subarticle with top 5 chunks

Experiments

Chunking splitting method







Baseline with splitting at 0,5 KB

Splitting at each article with top 3 chunks

Splitting at each subarticle with top 5 chunks

Key findings

Key findings

Small chunks (0.125KB - 0.5KB):

- Highest performance across all metrics
- Better context quality
- Improved answer relevance

Medium chunks (1KB - 4KB):

- Performance drop, especially in context quality
- Varied impact on faithfulness

Large chunks (8KB+):

- Lowest overall performance
- Poor context quality
- Lower answer correctness

Conclusion

Key findings

Small chunks (0.125KB - 0.5KB):

- Highest performance across all metrics
- Better context quality
- Improved answer relevance

Medium chunks (1KB - 4KB):

- Performance drop, especially in context quality
- Varied impact on faithfulness

Large chunks (8KB+):

- Lowest overall performance
- Poor context quality
- Lower answer correctness

Thank you for your attention!
Any questions?