Kunskapskontroll

Projekt i Data Science

Boris Lovrenovic

EC Utbildning

Projekt i Data Science

202602

Abstract

This report develops and evaluates a simple Retrieval-Augmented Generation (RAG) pipeline for context bound question answering. Web pages are collected via Google CSE, cleaned with trafilatura, chunked, embedded with Titan, and indexed in Qdrant for semantic retrieval. A generator LLM produces answers constrained by retrieved context, through a simple Streamlit UI. Two variants were compared on 10 fixed questions using a 1/0.5/0 scoring. The RAG variant with embeddings and vector search achieved a higher average score, indicating improved relevance over the baseline.

Innehållsförteckning

1 Inledning

Bakgrund: RAG i praktiska system och behovet av kontextbundna svar.

Syfte: utveckla och utvärdera en enkel RAG lösning.

Frågeställningar:

Hur påverkar RAG svarens relevans jämfört med en baslinje?

Vilka metodval ger bäst balans mellan kvalitet, kostnad och enkelhet?

2 Teori

RAG: retriever + generator, prompt som begränsar svar till kontext.

Embeddings + semantic search.

Chunking och dess påverkan på retrieval.

3 Metod

Identifiera & förbereda data

- Datakällor: webbsidor via Google CSE.
- Extraktion: text med trafilatura, borttagning av bilder/tabeller.
- Rensning: längdbegränsning + input sanering.
- Chunking: token baserat i chunk_utils.py.
- Embeddings: Titan via Bedrock i embedding_service.py.
- Vektordatabas: Qdrant i qdrant_service.py.

Metoder & tekniker

- RAG ger mer grundläggande svar än ren LLM-chatt.
- Embeddings + vektorsök behövs för semantic retrieval.
- Qdrant för snabb Nearest Neighbor sökning.
- LLM i generatorn för språkligt sammanfattade svar.

Modell

- E2E pipeline i rag_pipeline.py.
- UI i streamlit_app.py.

Agil metodik

Backlog:

- Sätta upp end-to-end pipeline för RAG (rag_pipeline.py)
- Implementera webbsök och URL-hämtning (search_service.py)
- Extrahera och rensa text (extractor.py, sanitization.py)
- Chunking av text och embeddings (chunk_utils.py, embedding_service.py)
- Vektordatabas och retrieval (qdrant_service.py)
- Generering med LLM (content_generator.py)
- Minimal UI (streamlit_app.py)
- Konfig/inställningar och .env (settings.py)

Sprint 1:

- Mål: få en enkel fråga-svar-kedja att fungera.
- Leverans: sök, extraktion, LLM-svar utan vektorsök.
- Justering: Begränsa textlängd

Sprint 2:

- Mål: lägga till embeddings och vektorsök.
- Leverans: chunking, embeddings, Qdrant-index och retrieval.
- Justering: byta prompt

Sprint 3:

- Mål: stabilisera pipeline och förbättra svarskvalitet.
- Leverans: input-sanering, retries, timeouts, förenklad UI.
- Justering: ändra chink_size / rag_top_k

4 Resultat och Diskussion

För att jämföra modell varianterna användes samma uppsättning frågor, 10st, där jag testade Variant A och Variant B.
Svar bedömdes med poängsättning 1 = korrekt, 0.5 = delvis korrekt, 0 = fel/avslag.

Variant A: SEARCH_LIMIT=1, RAG_TOP_K=1
Variant B: SEARCH_LIMIT_5, RAG_TOP_K=5

Frågor som ställdes:
What is the difference between a retriever and a generator in RAG?
Why is chunking used in RAG, and what is a downside of fixed-length chunking?
What are embeddings and how are they used in semantic search?
What is cosine similarity in short?
What is Qdrant and why is a vector database used in RAG?
What are the three pillars of Scrum?
What are the five Scrum events?
Which roles are included in a Scrum Team?
What is the difference between cloud-based and local language models?
When should a RAG model say "I don't know"?

Poängsättning för varje fråga

Variant A	Variant B	Notering
1	1	Korrekt skillnad retriever/generator
1	1	Korrekt om chunking + nackdel
0	1	A avstår, B korrekt
1	1	Korrekt definition
1	1	Korrekt om Qdrant + Vector DB
1	1	Korrekt
1	1	Korrekt
1	1	Korrekt
1	1	Korrekt
0	1	A avstår, B korrekt

Resultattabell

Modell	Summa	Snitt
Variant A	8	0.80
Variant B	10	1.00

5 Slutsatser

Syftet var att utveckla och utvärdera en enkel RAG lösning, vilket gjordes genom en E2E pipeline med webbsök, extraktion, chunking, embeddings, Qdrant retrieval och LLM.

Resultatet var att Variant B fick 10/10 poäng (snitt 1,00) jämfört med Variant A 8/10 (snitt 0,80).

Frågeställning 1 besvaras av att RAG förbättrar relevansen, eftersom varianten med retrieval gav fler korrekta svar än baslinjen.

Frågeställning 2 besvaras av att embeddings + vektorsök i Qdrant, med rimlig chunk storlek och top-k, gav bäst balans mellan kvalitet, kostnad och enkelhet.

Vald modell är Variant B eftersom den presterade bäst, men den begränsas beroende av webbkällor och chunk/prompt val.

En kritisk granskning är att lösningen bygger på webbkällor från Google CSE med låg SEARCH_LIMIT och att trafilatura tar bort tabeller/bilder samt trunkerar text till 5000 tecken, vilket kan ge viss bias eller tappad fakta.
Svaren saknar källhänvisningar eftersom generatorn bara returnerar text, och Qdrant collectionen rensas inte mellan körningar, så tidigare chunks kan läcka in.

Utvärderingen är manuell (10 frågor, poäng 1/0.5/0) utan automatiska tester eller reproducerbar loggning.
Kvaliteten är också känslig för “fasta chunks” (max_tokens=256, ingen overlap) och promptens hårda kontextkrav, där små ändringar i rag_top_k påverkar svaren.

Det agila arbetssättet med iterativa sprintar fungerade bra för att få en hel pipeline, men nästa gång skulle jag tidigare lägga in återställning/isolering av index, spårbarhet med källor och en tydligare plan.