A Retrieval Augmented Generation chatbot

for Tolkien lore

Kevin Lundstedt

NBI/Handelsakademin

AI-teori och tillämpning, del 2

202601

Abstract

This project explores the use of Retrieval-Augmented Generation (RAG) to build a chatbot that answers questions about Tolkien's works using a controlled set of source texts. Instead of allowing the language model to answer freely, the system retrieves relevant text chunks from a vector database and uses them as context for each response. The project focuses on understanding the full RAG pipeline, including data ingestion, embeddings, retrieval, and response generation. A working proof of concept was implemented using Python, OpenAI's API, Chroma as a vector database, and both a terminal-based interface and a Streamlit web application.

Innehållsförteckning

1 Inledning

Utvecklingen av stora språkmodeller har möjliggjort kraftfulla system för textgenerering och frågebesvarande. Samtidigt finns välkända utmaningar, särskilt risken för hallucinationer, där modellen genererar svar som låter trovärdiga men saknar stöd i faktiska källor. Detta är problematiskt i sammanhang där korrekthet, spårbarhet och transparens är viktiga.

Ett sätt att hantera detta är Retrieval-Augmented Generation (RAG), där språkmodellen kombineras med informationssökning i ett externt källmaterial. I stället för att svara fritt baserat på sin träningsdata hämtar modellen relevant kontext från en egen kunskapsbas och använder denna som underlag för sina svar. På så sätt kan svaren begränsas till ett definierat innehåll och kopplas till korrekta källor.

Detta arbete genomförs som en del av kursens kunskapskontroll inom AI – teori och tilläpning och fokuserar på praktisk implementering av en RAG-lösning. Projektet består av en chattbot med fokus på Tolkien-relaterat innehåll, där användaren kan ställa frågor och få svar som enbart baseras på ett eget textmaterial. Genom att bygga hela flödet från datainsamling till färdigt gränssnitt ges en helhetsförståelse för hur RAG-system fungerar i praktiken.

1.1 Syfte

Syftet med detta arbete är att undersöka hur en Retrieval-Augmented Generation-lösning kan implementeras i Python för att skapa en kontrollerad chattbot som endast svarar baserat på ett fördefinierat källmaterial. Målet är att förstå och demonstrera hela RAG-kedjan, från textdata och embeddings till retrieval och svarsgenerering, samt att minska risken för hallucinationer genom tydliga begränsningar och transparens.

1.2 Frågeställningar

För att uppfylla syftet besvarar arbetet följande frågeställningar:

1. Hur kan en RAG-pipeline byggas från grunden i Python med hjälp av embeddings och en vektordatabas?
2. Hur påverkar chunking, embeddings och retrieval kvaliteten och tillförlitligheten i svaren?
3. Hur kan källor och använd kontext synliggöras för att öka transparens och kontroll i ett RAG-system?

2 Teori

Detta kapitel presenterar den teoretiska bakgrunden som ligger till grund för arbetet. Fokus ligger på RAG, embeddings, vektordatabaser samt problematiken kring hallucinationer i stora

språkmodeller. Dessa begrepp är centrala för att förstå hur den utvecklade chattboten fungerar och varför arkitekturen valdes.

2.1 Retrieval Augmented Generation

Retrieval-Augmented Generation (RAG) är en arkitektur som kombinerar generativa språkmodeller med informationssökning i ett externt källmaterial. I stället för att modellen enbart genererar svar baserat på sin interna träningsdata, hämtas relevant kontext från en separat kunskapsbas vid varje fråga. Denna kontext används sedan som underlag när svaret genereras.

RAG-arkitekturen består av två delar: ett retrieval-steg och ett generation-steg. I retrieval-steget identifieras de mest relevanta textstyckena från en kunskapsbas baserat på användarens fråga. I generation-steget kombineras dessa textstycken med frågan i en prompt som skickas till språkmodellen. Resultatet blir svar som är mer faktabundna och spårbara än vid fri textgenerering.

Denna metod är särskilt användbar när man vill begränsa modellens svar till ett visst innehåll, exempelvis dokumentation, interna databaser eller Tolkien-relaterade texter.

2.2 Embeddings

Embeddings är numeriska representationer av text som fångar semantisk betydelse. Varje textstycke omvandlas till en vektor där liknande texter hamnar nära varandra. Detta möjliggör effektiv jämförelse mellan texter baserat på innebörd snarare än exakt ordmatchning.

I ett RAG-system används embeddings både för att representera källtexterna och för att representera användarens fråga. Genom att jämföra dessa vektorer kan systemet identifiera vilka textstycken som är mest relevanta för en given fråga. I detta projekt används embeddings från OpenAI, vilket möjliggör semantisk sökning över textmaterialet.

Kvaliteten på embeddings påverkar direkt kvaliteten på retrieval-steget. Om embeddings inte fångar textens innebörd på ett tillförlitligt sätt riskerar systemet att hämta irrelevanta eller missvisande textstycken.

2.3 Vektordatabaser

En vektordatabas är en databas som är optimerad för lagring och sökning av vektorer. Till skillnad från traditionella databaser, som ofta bygger på exakta matchningar eller indexering av text, möjliggör vektordatabaser snabb likhetsbaserad sökning.

I detta projekt används Chroma som vektordatabas. Varje textchunk lagras tillsammans med sin embedding och tillhörande metadata. När en fråga ställs beräknas dess embedding och de mest liknande vektorerna hämtas från databasen.

Användningen av metadata gör det möjligt att spåra exakt vilka källor som använts vid varje svar, vilket är centralt för transparens och verifierbarhet i RAG-system.

2.4 Hallucinationer och kontroll

Hallucinationer är ett välkänt problem i stora språkmodeller och innebär att modellen genererar information som inte stöds av faktiska källor. Detta kan ske även när svaret presenteras med hög språklig säkerhet. I sammanhang där korrekthet är viktig utgör detta en betydande risk.

RAG-arkitekturen är ett sätt att minska hallucinationer genom att begränsa modellens tillgång till information. Genom att endast tillåta svar baserade på hämtad kontext minskas modellens frihet att hitta på information. Ytterligare kontroll kan uppnås genom relevans-trösklar, där systemet avstår från att svara om ingen tillräckligt relevant kontext hittas.

I detta projekt instrueras modellen att tydligt ange när svaret inte finns i källmaterialet. På så sätt prioriteras korrekthet och transparens framför att alltid generera ett svar.

3 Metod

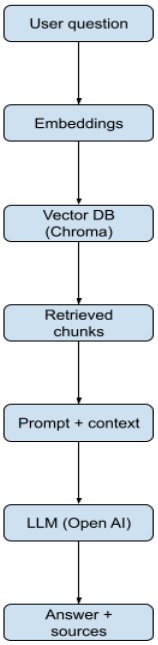
Detta kapitel beskriver hur projektet har genomförts i praktiken, från val av data och verktyg till implementation av RAG-lösningen. Fokus ligger på att redogöra för arbetsprocessen och de tekniska val som gjorts snarare än på detaljerad kodgenomgång.

3.1 Översikt av systemarkitekuren

Projektet är uppbyggt kring en tydlig separation mellan datainsamling, indexering, retrieval-logik och användargränssnitt. Arkitekturen består av fyra huvudsakliga delar:

- Ett ingest-steg som förbereder och indexerar textdata
- En central RAG-modul som hanterar retrieval och svarsgenerering
- Ett terminalbaserat gränssnitt för testning
- Ett webbaserat gränssnitt byggt med Streamlit

Genom att samla all RAG-logik i en gemensam modul kunde samma funktionalitet återanvändas i flera gränssnitt utan kodduplicering.



Figur 1. Översikt av RAG-flödet som används i projektet.

3.2 Indexering och embeddings

Textmaterialet samlades in manuellt från Tolkien-relaterade Wikipedia-sidor. Texterna sparades som separata .txt-filer. Detta val gjordes för att få sammanhängande, beskrivande text lämpad för semantisk sökning, till skillnad från tabellbaserade dataset som ofta används i andra sammanhang.

Innan indexering kontrollerades att varje textfil innehöll relevant innehåll och tomma filer exkluderades. Varje dokument försågs med metadata, bland annat källfilens namn och en titel baserad på filnamnet, för att underlätta spårbarhet vid retrieval.

3.3 Indexering och embeddings (ingest-steg)

Indexeringen genomfördes i ett separat ingest-steg som körs fristående från själva chattbotten. I detta steg delades varje dokument upp i mindre textstycken, så kallade chunks.

För varje chunk skapades embeddings med hjälp av OpenAI:s embedding-modell. Dessa embeddings lagrades tillsammans med tillhörande metadata i en vektordatabas (Chroma). Varje chunk tilldelades även ett unikt chunk-ID för att säkerställa stabila referenser vid retrieval.

Syftet med ingest-steget är att omvandla rå text till en struktur som möjliggör effektiv semantisk sökning.

3.4 Retrieval-Augmented Generation-logik

När en användare ställer en fråga aktiveras den centrala RAG-funktionen. Först omvandlas frågan till en embedding som jämförs med de embeddings som lagrats i vektordatabasen. De mest relevanta chunksen hämtas baserat på semantisk likhet.

De hämtade textstyckena används sedan som kontext i en prompt som skickas till språkmodellen. Prompten innehåller tydliga instruktioner som begränsar modellen till att endast svara baserat på den tillhandahållna kontexten. Om relevansen på de hämtade chunksen understiger en fördefinierad tröskel returneras i stället ett svar som indikerar att informationen inte finns i källmaterialet.

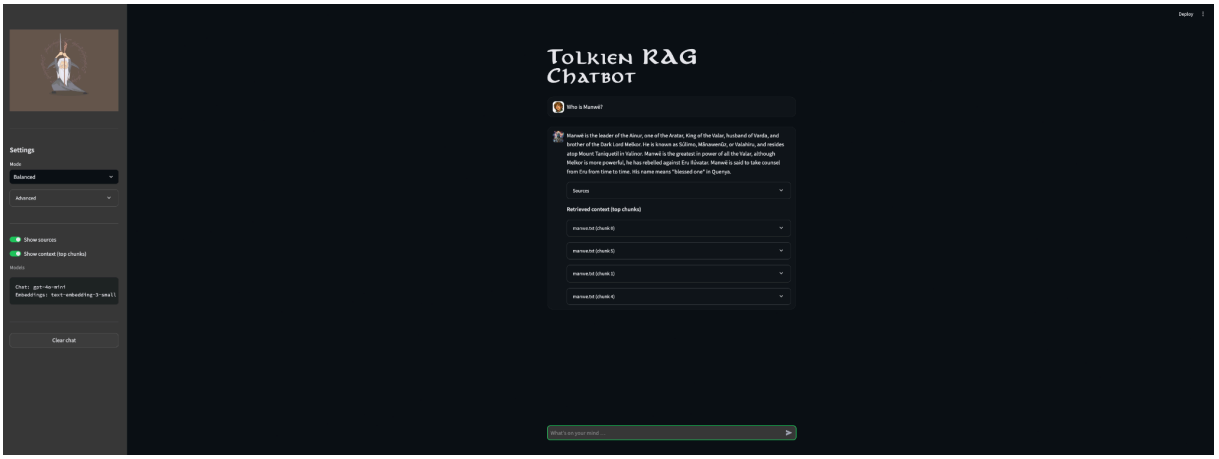
Denna logik säkerställer att modellen inte svarar fritt och minskar risken för hallucinationer.

3.5 Användargränssnitt

Två olika gränssnitt implementerades för att interagera med RAG-systemet. Det terminalbaserade gränssnittet erbjuder ett enkelt sätt att testa funktionaliteten och följa flödet från fråga till svar. Streamlit-applikationen bygger på samma logik men presenterar svaren visuellt, med möjlighet att visa källor och använd kontext.

Båda gränssnitten anropar samma RAG-funktion, vilket gör lösningen modulär och lätt att vidareutveckla med fler gränssnitt vid behov.

Figur 1. Terminalbaserat gränssnitt för RAG-chatbotten.



Figur 2. Webbaserat användargränssnitt byggt med Streamlit.

4 Resultat och Diskussion

Resultatet av projektet är en fungerande proof of concept som demonstrerar hela Retrieval-Augmented Generation-kedjan, från rå textdata till genererade svar baserade på hämtad kontext. Systemet kan svara på frågor om Tolkiens värld när relevant information finns i källmaterialet och avstår från att svara när tillräckligt stöd saknas.

4.1 Resultat

Den implementerade RAG-lösningen visar att det är möjligt att bygga en kontrollerad chatbot där språkmodellen inte svarar fritt, utan begränsas till ett eget textmaterial. Vid frågor som tydligt matchar innehållet i vektordatabasen hämtas relevanta text-chunks och används som kontext för svaret. Detta resulterar i faktabundna svar som kan kopplas till specifika källor.

Systemet klarar även enklare följdfrågor genom att hålla reda på tidigare ämnen i konversationen. Detta gör interaktionen mer naturlig och visar hur RAG kan kombineras med basic kontextmedvetenhet.

I Streamlit-gränssnittet visas både källor och, vid behov, de textstycken som faktiskt användes vid retrieval. Detta ökar transparensen och gör det tydligt hur svaret har genererats.

4.2 Diskussion

Ett tydligt resultat är att kvaliteten på svaren i hög grad beror på kvaliteten och strukturen på källmaterialet. Wikipedia-texterna fungerar väl för faktabaserade frågor, men är begränsade när det gäller mer djupgående eller tolkande frågor. Detta visar att RAG inte ersätter behovet av bra data, utan snarare förstärker värdet av den.

Valet av storlek på chunks och överlapp har haft stor påverkan på retrieval-kvaliteten. För stora chunks riskerar att innehålla för mycket irrelevant information, medan för små chunks kan leda till att sammanhang går förlorat. Dessa parametrar krävde praktisk tuning snarare än en statisk lösning.

Relevans-tröskeln visade sig vara ett effektivt sätt att minska hallucinationer. Genom att låta systemet avstå från att svara när stödet i källorna är svagt ökar tillförlitligheten, men detta sker på bekostnad av att vissa frågor lämnas obesvarade. Detta är dock ett medvetet designval i linje med projektets syfte.

4.3 Begränsningar

Projektet är begränsat till ett relativt litet och manuellt insamlat textmaterial. Detta påverkar både täckning och variationsrikedom i svaren. Vidare används en enkel “tumregel” för språkidentifiering och följdfrågor, vilket fungerar i många fall men inte är fullständigt robust.

Systemet saknar även en formell evalueringsmetod för retrieval-kvalitet, vilket gör bedömningen av resultatet lite subjektivt.

5 Slutsatser

5.1 Svar på frågeställningar

Den första frågeställningen rörde hur en Retrieval-Augmented Generation-pipeline kan byggas från grunden i Python. Arbetet visar att detta kan uppnås genom att kombinera ett ingest-steg för textdata, embeddings för semantisk representation och en vektordatabas för informationssökning. Genom att separera indexering, retrieval och svarsgenerering skapades en tydlig arkitektur där varje del har ett avgränsat ansvar.

Den andra frågeställningen handlade om hur chunking, embeddings och retrieval påverkar kvaliteten på svaren. Resultatet visar att dessa komponenter har stor inverkan på både precision och tillförlitlighet. Chunk-storlek och överlapp kräver praktisk justering för att balansera sammanhang och träffsäkerhet, medan embeddings-modellens kvalitet avgör hur väl semantisk likhet fångas. Retrieval-steget visade sig vara den mest kritiska delen, eftersom felaktigt hämtad kontext direkt leder till sämre eller missvisande svar.

Den tredje frågeställningen fokuserade på transparens och kontroll. Genom att visa vilka källor och textstycken som använts vid varje svar, samt genom att införa en relevans-tröskel, blev systemets beteende mer förutsägbart och spårbart. Detta ökade både användarens förståelse för svaren och minskade risken för hallucinationer.

5.2 Sammanfattande slutsats

Sammanfattningsvis visar projektet att RAG är en effektiv metod för att bygga mer kontrollerade och tillförlitliga chattbotar. Genom att begränsa språkmodellen till ett eget källmaterial och tydligt separera retrieval från generering kan risken för hallucinationer reduceras avsevärt. Lösningen fungerar väl som proof of concept och ger en god praktisk förståelse för hur RAG-system kan byggas, konfigureras och användas i verkliga tillämpningar.

6 Teoretiska frågor

Vad innebär Retrieval-Augmented Generation (RAG)?

Retrieval-Augmented Generation är en metod där en språkmodell kombineras med informationssökning. I stället för att svara enbart utifrån sin interna kunskap hämtar modellen relevant kontext från en extern källa, exempelvis en vektordatabas, och använder denna som grund för sitt svar.

Vad är embeddings och varför används de?

Embeddings är numeriska representationer av text som fångar semantisk betydelse. De används för att möjliggöra likhetsjämförelser mellan frågor och textstycken, vilket gör det möjligt att hitta relevanta delar av ett större textmaterial.

Vad är en vektordatabas?

En vektordatabas lagrar embeddings och gör det möjligt att snabbt söka efter de textstycken som är mest lika en given fråga. I detta projekt används Chroma för att lagra chunks tillsammans med metadata.

Varför delas texten upp i chunks?

Text delas upp i mindre delar för att öka precisionen vid retrieval. Mindre chunks gör det lättare att hitta exakt relevant information, medan överlapp används för att undvika att viktig kontext bryts mellan chunk-gränser.

Hur minskar RAG risken för hallucinationer?

Genom att begränsa modellen till att endast svara utifrån hämtad kontext minskar risken att modellen hittar på information. En relevans-tröskel används för att avgöra om tillräckligt stöd finns i källmaterialet.

7 Självreflektion

Vad har varit mest intressant i kunskapskontrollen?

Det mest intressanta har varit att bygga hela RAG-flödet från grunden och se hur de olika delarna samverkar i praktiken. Särskilt givande var att förstå hur embeddings och vektordatabaser används för att styra en språkmodell och hur tydlig kontext kan minska risken för hallucinationer. Att visuellt kunna följa vilka textstycken som faktiskt används i svarsgenereringen gav en djupare förståelse för hur RAG fungerar bortom teori.

Vad har varit mest utmanande i arbetet och hur har du hanterat det?

Den största utmaningen var att få retrieval-steget stabilt och tillförlitligt. Detta krävde experimenterande med chunk-storlek, relevans-trösklar och förståelse för hur man promptar. Utmaningen hanterades genom iterativ testning och genom att införa tydliga regler för när modellen ska avstå från att svara. Även strukturingen av projektet så att samma logik kunde användas i både terminal- och webbgränssnitt krävde eftertanke men resulterade i en renare arkitektur.

Om du skulle fortsätta utveckla din kompetens inom detta område vad skulle du fokusera på och varför?

Om jag skulle fortsätta utveckla min kompetens inom detta område skulle jag fokusera på att fördjupa min förståelse för hur RAG-system kan optimeras för bättre svarskvalitet. Det gäller särskilt val av chunk-strategier, embedding-modeller och relevansfiltrering, då dessa visade sig ha stor påverkan på systemets resultat.

Appendix A

Github-repository:

https://github.com/ecthelionofthefountain-510/kunskapskontroll-ai-theory-2

Streamlit:

Streamlit-applikationen finns deployad via Streamlit Cloud och kan köras direkt från webbläsaren.

https://ecthelionofthefountai-tolkien-rag-chatbotsrcstreamlit-xos6xz.streamlit.app/

Reproducrebarhet:

All kod för datainsamling, indexering, RAG-logik samt gränssnitt finns i Github-repot. Projektet kan reproduceras genom att följa instruktionerna i README-filen.

Källförteckning

Wikipedia – *Tolkien-relaterade artiklar (insamlade som textmaterial):*

https://lotr.fandom.com/wiki/Main_Page

OpenAI – *API-dokumentation:*

https://platform.openai.com/docs

Chroma – *Vector Database:*

https://www.trychroma.com

LangChain – *Documentation:*

https://python.langchain.com

Föreläsningsmaterial från NBI/Handelsakademin, kursen ***AI – teori och tillämpning***

Boken ***AI från grunden - tillämpad maskininlärning med Python***