

Digital Doktor

Retrieval-Augmented Generation av VGR Riktlinjer

Examensarbete inom högskoleprogrammet Datateknik högskoleingenjör (180hp)

Alice Emanuelsson & Adam Persson

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025
www.chalmers.se

EXAMENSARBETE 2025

Digital Doktor
Retrieval-Augmented Generation av VGR
Riktlinjer

Alice Emanuelsson & Adam Persson



CHALMERS

Institutionen för Data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025

Digital Doktor: Retrieval-Augmented Generation av VGR Riktlinjer
Alice Emanuelsson & Adam Persson

© ALICE EMANUELSSON, ADAM PERSSON , 2025.

Handledare: Araz Rawshani, Sahlgrenska Universitetssjukhuset
Handledare: Sakib SisteK, Institutionen för Data- och informationsteknik
Examinator: Nicholas Smallbone, Institutionen för Data- och informationsteknik

Examensarbete 2025
Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Överblick av programmets olika delar.

Skriven i L^AT_EX
Göteborg 2025

Digital Doktor: Retrieval-Augmented Generation av VGR Riktlinjer
ALICE EMANUELSSON, ADAM PERSSON
Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

Idag har Large Language Models (LLM) blivit en naturlig del i vardagen för många, då tillgängligheten och träffsäkerhet ökat markant de senaste åren. Det blir då en naturlig utveckling att skapa möjligheter för att använda LLM:s som stödverktyg inom vården. Detta examensarbete görs i samarbete med Sahlgrenska Universitets Sjukhus och syftar till att utveckla Retrieval-Augmented Generation baserat på Västra Götalandsregionens medicinska riktlinjer. Med hjälp av OpenAIs GPT API och riktlinjerna för regionen, skapas ett program med chatt funktionalitet som levererar svar på medicinska frågor baserat på aktuella riktlinjer, samt tydligt källhänvisar till dessa. Projekt avser att visa på potentialen av dessa program. Rapporten redovisar hur arbetet genomfördes och användbarheten av programmet inom vården.

Nyckelord: RAG, LLM, OpenAI, Azure, python, vådriktlinjer

Förord

Examensarbetet har utförts på kandidat nivå vid institutionen för Data- och informationsteknik på Chalmers Tekniska Högskola. Arbetet utfördes i samarbete med Sahlgrenska Universitets Sjukhuset i Göteborg och vi vill rikta ett stort tack till vår handledare på sjukhuset, Araz Rawshani för att vi fick möjligheten att utföra detta examensarbete. Vi vill även tacka för all handledning och för utvärderingen av produkten. Vi önskar också att tacka Sakib Sisteck vår handledare på Chalmers Tekniska Högskola, för allt stöd i samband med vårt projekt. Tusen tack!

Alice Emanuelsson & Adam Persson, Göteborg, Maj 2025

Beteckningar

Här är en lista av beteckningar som används i rapporten samt vad de står för.

AI	Artificiell intelligens
API	applikationsprogrammeringsgränssnitt
CSS	Cascading Style Sheets
GPT	Generative pre-trained transformer
HTML	Hyper Text Markup Language
LLM	Large Language Model
PDF	Portable Document Format
URL	Uniform Resource Locator
UX	User Experience
VGR	Västra Götalandsregionen

Innehåll

Akronymer	ix
Figurer	xiii
1 Inledning	1
1.1 Bakgrund	1
1.2 Syfte	1
1.3 Mål	1
1.4 Avgränsningar	2
2 Metod	3
2.1 Informationsinsamling	3
2.2 Agil metodik	3
2.3 Verifiering av resultat	3
3 Teknisk Bakgrund	5
3.1 Frontend	5
3.1.1 Django	5
3.1.2 HTML, CSS & JavaScript	5
3.1.3 Bootstrap	6
3.2 Backend	6
3.2.1 Python	6
3.2.2 LLM	6
3.2.3 Lokalt under utveckling	6
3.2.3.1 Ollama & LLaMA 3.2 (1B)	7
3.2.3.2 Inbäddning med Sentence Transformers	7
3.2.4 Molntjänster under distribution	7
3.2.4.1 GPT-4o-mini	7
3.2.4.2 Inbäddning med OpenAI	7
3.2.5 Retrieval-Augmented Generation	8
3.2.6 Azure & Github	8
3.2.7 Webbskrapning & BeautifulSoup	8
3.3 Övriga Python-bibliotek	8
4 Genomförande	11
4.1 Utvecklingsfas av webbapplikation	11
4.1.1 Design	11

4.1.2	Bygga webbapplikationen	12
4.1.3	Hämta ned riktlinjer från Västra Götalandsregionens webbsida	13
4.1.4	Bearbetning av PDF:er	14
4.1.5	Första versionen av inbäddningar & sökande efter relevanta textbitar i datan	14
4.1.6	Formulera uppdaterad fråga med kontext till LLM	15
4.1.7	Testar RAG med llma3	16
4.2	Distrubitionsfas	16
4.2.1	Koppla LLM API till projektet	16
4.2.2	Publicera webbapplikation	17
4.3	Vidareutvecklingsfas	17
4.3.1	Testa första modellen	17
4.3.2	Undersöka förbättringsområden	18
4.3.3	Andra versionen av inbäddningar & sökande efter relevanta textbitar i datan	18
4.4	Testning & justerings fas	18
4.4.1	Initiala tester och justeringar	18
4.4.2	Omfattande testning	20
5	Resultat	21
5.1	Utseende	21
5.2	Svaren från programmet	22
5.3	Historik	23
5.4	Kvalitativ analys av svar	23
5.5	Systemkonstruktion	25
6	Slutsats	29
7	Diskussion	31
7.1	Utvecklingsmöjligheter	31
7.1.1	Ekologiska aspekter	32
7.1.2	Etiska & samhällsmässiga aspekter	33
	Bibliography	35

Figurer

4.1	Figuren visar designalternativet för webbapplikation som var baserat på pratbubblor.	12
4.2	Figuren visar designalternativet för webbapplikation som var baserat på svar i textbox.	13
5.1	Figuren visar den slutgiltiga designen av webbapplikationen.	22
5.2	Figuren visar hur den responsiva designen fungerar i den slutgiltiga versionen av webbapplikationen, där layouten ändras från rad- till kolumnbaserad när skärmbredden minskar.	22
5.3	Figuren visar övergripande schema över hur programmet hämtar ned riktlinjer och bearbetar dessa för användning vid körning.	26
5.4	Figuren visar övergripande schema över hur programmet fungerar under körning, med fokus på frontend-funktionalitet.	26
5.5	Figuren visar hur programmet fungerar under körning, med en övergripande översikt av funktionaliteten i backend.	27
7.1	Skillnad i koldioxidutsläpp per producerad kilowattimme el mellan olika länder [1].	32

1

Inledning

I denna del av rapporten beskrivs bakgrunden till projektet samt preliminärt syfte, mål och avgränsningarna. Här finner läsaren en överblick av vad rapporten handlar om.

1.1 Bakgrund

En av de största riskerna inom sjukvården är den mänskliga faktorn. År 2000 uppskattades det att 98 000 människor dog som direkt resultat av mänskliga misstag av läkare och annan vårdpersonal på sjukhus bara i USA. Detta var mer än dubbelt så många som bilolyckor och även bröstcancer, vid tillfället uppskattningen gjordes. [2] I strävan efter säkrare vård blir det därför ett naturligt steg att vilja minska den mänskliga faktorn. Artificiell intelligens (AI) har gjort stora framsteg de senaste åren och kan bli ett stort stöd i beslutsfattanden inom sjukvården i framtiden. För att AI ska användas på ett smidigt sätt inom vården måste det göras enkelt att använda med referenser till vårdens riktlinjer[3], så att vårdpersonal får den information de behöver för att kunna fatta beslut som påverkar patienter.

1.2 Syfte

Syftet med projektet är att visa på hur AI kan agera som stöd för att minska den mänskliga faktorn inom vården, när det kommer till vanliga frågor som idag ställs till läkare. Informationen ska presenteras på ett sådant vis att det blir enklare för vårpersonal att lita på information eftersom den ger referenser till utstakade riktlinjer de ska följa.

1.3 Mål

Målet med detta projekt är att utveckla en säker och tillgänglig webbapplikation för AI-baserad hälso- och sjukvårdsrådgivning. Applikationen ska vara anpassad för sjukvårds personal genom att ha tydliga referenser till riktlinjer för regionerna. Webbapplikationen kommer inte förväntas vara en färdig produkt i slutet av projektet utan en enklare version av vad som kan förväntas av en sådan webbapplikation.

- Webbapplikation uppbyggd i programmeringsspråket Python
- Användning av AI applikationsprogrammeringsgränssnitt (API)

- Anpassa svaren med Retrieval-Augmented Generation (RAG) för sjukvårdens riktlinjer

1.4 Avgränsningar

För att säkerställa genomförbarheten inom projektets tidsram görs följande avgränsningar:

- Fokus kommer att ligga på språkförståelse och teknisk implementation snarare än medicinsk diagnostik.
- Systemet kommer inte att inkludera en fullständig elektronisk journalhantering utan endast hantera temporära konversationer.
- Applikationen kommer inte ha som mål att ersätta professionell medicinsk rådgivning utan enbart fungera som ett stödverktyg.
- Det kommer vara en begränsad mängd riktlinjer som hanteras i projektet.

2

Metod

I detta kapitel presenteras metoderna som ställts upp inför projektet.

2.1 Informationsinsamling

Eftersom det var ett programmeringsbaserat projekt samlades stora mängder av information in från dokumentationer på internet, men även Youtube videor och forum på internet användes som stöd under projektets gång. Då projektet byggde på ramverk och verktyg som vi inte hade använt tidigare spenderades första delen av projektet på att bekanta sig med dessa. Även akademiska artiklar användes i projektrapporten och dessa hittades huvudsakligen på de olika databaserna som finns tillgängliga på Chalmers bibliotek.

2.2 Agil metodik

För att nå målet med projektet tillämpades agilmetoder för att öka flexibilitet och att organisera upp arbetet. Agila metoder är lämpliga för denna typ av projekt då det i förväg är svårt att ha en tydlig översikt över hur lång tid olika delar av projektet kan förväntas ta [4]. Speciellt vid användning av nya frameworks och verktyg som vi inte tidigare arbetat med. Metoder som till exempel waterfall kan bli svåra att tillämpa utan tidigare erfarenheter eller studier om hur lång tid projekt av denna typen tar. Waterfall baseras nämligen på olika stadier som är fasta och väldefinierade [4]. En nackdel med den agila metoden är att det inte blir överblickbart på ett enkelt sätt, när element kan förväntas vara klara. Det är enklare i waterfall principen. Målet var att ha veckovisa sprintar där det med hjälp av scrummetodiken går att organisera upp arbetet. Projektet delades in i mindre delmål och som kundes planeras utefter den beräknade tidsåtgången för varje moment. Kanbanbord användes för att medföra en större synlighet i projekt, och därmed ge större överblick över vad som behövdes göras. Nästa del att göra i projektet kunde därför avgöras smidigare.

2.3 Verifiering av resultat

Programmering utgör en central del av projektet, och därför kommer ett stort fokus att ligga på trial-and-error. Inom programmering är det vanligt att iterativt testa olika lösningar för att successivt närma sig en fungerande implementation. Denna

metod präglar programmeringsarbetet i projektet, särskilt utanför AI-delen, där lösningar utvecklas genom upprepade tester och justeringar.[5]

När det gäller svaren från AI-modellen skiljer sig tillvägagångssättet, eftersom dessa är genererade snarare än förutbestämda. För att verifiera korrektheten hos AI-modellens svar, som är inriktade på medicinsk konsultation, används en kombination av kvalitativa och kvantitativa metoder. Den kvalitativa delen innebär att en läkare manuellt granskar ett urval av AI-svar och bedömer hur väl de stämmer överens med de regionala rekommendationerna, med fokus på medicinsk relevans och noggrannhet.

Den kvantitativa delen görs under utveckling för att undersöka om programmet fungerar som tänkt. Kvantitativa frågor hämtas genom manuell läsning av PDF:er och så ställs frågorna till programmet. Sedan avgörs det om frågans svar hänvisar till rätt dokument, samt om svaret kan anses rimligt. Alternativt används LLM modeller som exempelvis NotebookLM för att hitta rimliga frågor baserat på PDF:er. Till viss del jämförs även svaren för att mäta semantisk likhet.

För att utvärdera lösningen kommer vi att besvara ett antal centrala frågor:

- Hur väl stämmer AI:s svar överens med medicinska riktlinjer?
- Hur ofta behöver läkare korrigera AI:s rekommendationer?
- Hur användbar upplever vårdpersonal AI-lösningen i praktiken?
- Går det att minska antalet felbeslut med hjälp av AI?

En lyckad lösning innebär att programmet tillhandahåller relevanta och pålitliga svar, minskar arbetsbelastningen för vårdpersonal och bidrar till säkrare beslut. Detta bedöms genom en kombination av testresultat och feedback från personal med medicinsk expertis.

3

Teknisk Bakgrund

I den tekniska bakgrunden presenteras de olika verktygen och ramverk som används i projektet.

3.1 Frontend

I denna del av den tekniska bakgrunden presenteras de delar som är relevanta för byggandet av frontend i projektet.

3.1.1 Django

Django är ett högnivå Python ramverk enligt Django Software Foundation [6]. Ramverket var publicerat 2005 av Adrian Holovaty och Simon Willson, men ägs nu av Django Software Foundation [7]. Django använder sig av tydliga design principer som bidrar till att göra ramverket produktivt och skalbart. Exempelvis används Don't Repeat Yourself (DRY) principen för att göra programmen enklare genom att undvika upprepning och skapa standarder i applikationen[7]. Alltså räcker det att definiera en gång hur till exempel ett speciellt typ av fält ska se ut, vilket sen kan refereras till på flera ställen i applikationen. Mallar är en stor del av Django utveckling och är huvudsakligen uppbyggda i Hyper Text Mark Up Language (HTML). Projekt i Django har ofta mappar med de olika mallarna som används i programmet [7]. Django gör det lätt att bygga en webbapplication med stöd för URL:er.

3.1.2 HTML, CSS & JavaScript

Hyper Text Markup Language, HTML som kort är ett verktyg för att formatera information på webbsidor. Med hjälp av taggar går det att bestämma vilken typ av format informationen ska visas som.[8] Det går alltså att dela upp sidan i delar, skapa klasser, knappar, texturutor med mera. HTML används ofta tillsammans med Cascading Style Sheets (CSS), som är till för att visuellt formatera HTML. Att ändra på den visuella delen av en webbsida är ofta betydligt lättare i CSS än i HTML, även om det går att göra i endast HTML [9]. JavaScript är ett skriptspråk och inte ett helt programmeringsspråk, som ofta används vid webbutveckling[10]. Det används för att göra webbsidorna interaktiva. Språket är även dynamiskt och lätt[11]. Att använda JavaScript för klient sidan är det vanligaste alternativet, och det ska finnas referenser till skriptet i HTML sidor vid användning, alternativt att hala skriptet återfinns där[11].

3.1.3 Bootstrap

Bootstrap är ett populärt frontend verktyg som fungerar på alla stora plattformar och sökmotorer enligt deras dokumentation [12]. En fördel med Bootstrap verktyget är att det baseras på att normalisera utseendet av applikationen så att det ska se likadant ut oberoende av vilken webbläsare som används samt att den går att göra responsiv för olika plattformar. Den utvecklades först och främst för användning på telefoner, men att det enkelt går att justera för användning på dator och andra enheter. [12]. Genom användning av Bootstraps egna komponenter och hantering av layout effektiviseras alltså skapandet av webbsidor som kan fungera för flera olika plattformar och sökmotorer. För användning av Bootstrap skapas klasser som innehåller deras färdiga mallar. Det går att bygga på klassnamnen med olika komponenter tills den uppfyller kraven. De olika komponenterna finns beskrivna i dess dokumentation där det även finns tydliga exempel [12].

3.2 Backend

I denna del av den tekniska bakgrunden presenteras relevant information om de olika språk och verktyg som avser backend utvecklingen av projektet.

3.2.1 Python

Python är ett högnivå programmeringsspråk som för publicerades för allmänheten 1991. Python är ett väldigt kraftfullt programmeringsspråk som även har ett tydligt syntax som har likheter till naturligt språk. [13] Språket är i stor del uppbyggd i olika bibliotek som kan användas för väldigt specifika och komplicerade beräkningar på ett enkelt vis[14]. Med sina många högnivå bibliotek anses språket vara välanpassat till användning inom artificiell intelligens (AI), flexibelt och enkelt att använda.[14] Eftersom Python och dess bibliotek är så bra anpassat för användning av LLMs så är det det språket som kommer att användas i projektet.

3.2.2 LLM

Large Language Models är generativa AI-program som använder naturligt språk som in och utdata[15]. En LLM fungerar genom att omvandla indata till tokenar som används för att göra om text till sekvenser. Dessa sekvenser kan sedan användas för att skapa inbäddning. Med hjälp av token vikter och inbäddningar kommer modellen att försöka förutspå nästa token[15]. På det viset försöker modellen hitta det svar som den förutspår passar frågan bäst. För att träna an stor LLM krävs alltså stora mängder data, då den ska förutspå svaret på mängder av olika frågor.

3.2.3 Lokalt under utveckling

Här presenteras några av de verktyg som användes i utvecklings fasen av projektet men som inte används i den färdiga produkten.

3.2.3.1 Ollama & LLaMA 3.2 (1B)

Ollama är en plattform för lokalt körbara språkmodeller (LLM:er) som erbjuder ett smidigt och resurssnålt sätt att exekvera avancerade AI-modeller utan beroende av molntjänster. Genom att paketera modeller i körbara miljöer möjliggör Ollama effektiv och säker inferens direkt på användarens maskin, vilket är särskilt värdefullt i scenarier där datasekretess, svarstid eller nätverksberoende är prioriterat. I detta projekt används Ollama för att köra Meta's LLaMA 3.2-modell med 1 miljard parametrar, en kompakt men kraftfull språkmodell som erbjuder en god balans mellan prestanda och resursförbrukning. LLaMA 3.2 (1B) är optimerad för att leverera högkvalitativa textgenereringar även på hårdvara med begränsad kapacitet, vilket gör den idealisk för prototypning och lokal testning under utvecklingsfasen[16].

3.2.3.2 Inbäddning med Sentence Transformers

För att representera text på ett numeriskt sätt användes biblioteket Sentence Transformers i Python. Den specifika modellen som valdes var "all-MiniLM-L6-v2", som är väl lämpad för att skapa semantiska inbäddningar av meningar och kortare paragrafer [17]. Dessa inbäddningar möjliggör effektiv jämförelse av semantiskt innehåll, vilket är avgörande för att exempelvis kunna matcha frågor med relevanta svar eller dokument. Som i uppgiftens fall är dokument från VGR:s riktlinjer.

3.2.4 Molntjänster under distribution

Här presenteras de tjänster som applicerades i distributionsfasen av projektet.

3.2.4.1 GPT-4o-mini

Generative pre-trained transformer (GPT) är företaget OpenAIs stora språkmodeller[18]. De har flera LLM var av GPT-4o-mini är en av de LLM APIn som går att använda[19]. GPT är avancerad språkmodeller utvecklade för att hantera komplexa naturliga språkuppgifter med hög precision. De är en typ av LLM, med särskilt fokus på språkgenerering. Den är snabb och billigare att använda i jämförelse med GPT-4 och tidigare versioner eftersom att den gör mer fokuserade uppgifter[20]. Den är en de modeller som är anpassade för att hantera finjustering[20].

3.2.4.2 Inbäddning med OpenAI

Inbäddningar är numeriska representationer av text som används för att identifiera relationer och likheter mellan olika textstycken [21]. Detta gör det möjligt för stora språkmodeller (LLM) att förstå och arbeta med språkliga samband.

Vi inledde arbetet med modellen "all-MiniLM-L6-v2" från Sentence Transformers, som är snabb och resurssnål och fungerar bra för allmän semantisk jämförelse. Under projektets gång bytte vi dock till OpenAIs modell "text-embedding-ada-002", eftersom den visade sig ge bättre resultat i kombination med OpenAIs egna språkmodell.

Teoretiskt sett är "text-embedding-ada-002" bättre lämpad för denna typ av applikation eftersom den är tränad på betydligt större och mer diversifierade mängder

data, är optimerad för semantisk sökning och kontextuell förståelse, samt är direkt kompatibel med OpenAIs infrastruktur. Detta gör att den kan generera mer informativa och finfördelade vektorer, vilket i sin tur förbättrar precisionen vid matchning av användarfrågor mot innehållet i VGR:S riktlinjer.

3.2.5 Retrieval-Augmented Generation

För LLM som ska svara på specifika frågor inom ett specifikt ämne kan det vara lämpligt att använda Retrieval-Augmented Generation (RAG). Det går att anpassa LLM:ens svar genom att ge den relevant data som den ska basera svaren på. Svaren kan även ge referenser till de källor där informationen hämtats för att öka pålitligheten[22]. Alltså kan de stora LLM vi är vana vid anpassas till vårt behov.

RAG baseras på att frågan som ställs till LLM modellen ska först hanteras med hjälp av en inbäddningsmodell för att hitta relevant information i data som svaren ska baseras på. LLM modellen får sedan denna justerade fråga som innehåller denna kontext och kan därmed ge bättre anpassade svar [22].

3.2.6 Azure & Github

Azure är en populär molnplattform av Microsoft[23]. I detta projekt används det för distribution och hosting av webbapplikation samt hantering av openAIs API tjänster. Github är en tjänst för lagring av GIT-projekt [24]. Github användes för versionshantering och lagring av koden, samt att det underlättade samarbetet i projektet.

3.2.7 Webbskrapning & BeautifulSoup

Webbskrapning är en teknik för att extrahera information från webbsidor, vilket är särskilt användbart när data inte finns tillgänglig genom traditionella API:er.[25] I detta projekt används webbskrapning för att samla in relevant textdata från publika webbplatser, vilket sedan används som underlag för vidare analys och träning av språkmodeller. Ett av de centrala verktygen för denna uppgift är BeautifulSoup. BeautifulSoup är ett Python-bibliotek som möjliggör enkel parsning och navigering av HTML- och XML-dokument. Genom att kombinera BeautifulSoup med verktyg som Requests för HTTP-förfrågningar, kan specifika delar av en webbsida som exempelvis artiklar, rubriker eller metadata extraheras med hög precision. Denna metod är särskilt effektiv när strukturen på webbsidan är relativt stabil, vilket gör det möjligt att automatisera insamlingen av stora mängder ostrukturerad text. Den insamlade datan förbehandlas därefter för att integreras som input för språkmodeller [25].

3.3 Övriga Python-bibliotek

Förutom de ramverk och bibliotek som beskrivits tidigare (såsom **Django**, LLM-integrationer och **BeautifulSoup**), används ett antal kompletterande Python-bibliotek

som stödjer projektets funktionalitet på olika nivåer. Dessa kan delas in i följande kategorier:

Databashantering och kopplingar

- **psycopg2** används för att koppla Django-applikationen till en PostgreSQL-databas. Detta möjliggör lagring och hämtning av användardata, såsom tidigare chattkonversationer, användarprofiler och sessionsdata [26].
- **dj-database-url** används för att läsa databasinställningar från miljövariabler, vilket förenklar konfigurationen i olika miljöer (lokalt, staging, produktion) [27].

Databearbetning och vetenskapliga beräkningar

- **Pandas**, **NumPy**, **SciPy** används för databehandling, statistisk analys och numeriska beräkningar [28, 29, 30]. Dessa bibliotek är centrala för att strukturera och analysera data på ett effektivt sätt.

Maskininlärning och vektorsökning

- **Scikit-learn** och **Faiss** används för likhetsmätning, vektorsökning och enklare maskininlärningsalgoritmer, särskilt inom ramen för RAG-arkitekturen [31, 32].
- **Joblib** och **ThreadPoolctl** används för att förbättra prestanda vid parallellisering av vissa ML-uppgifter [33, 34].

Kommunikation och API-integrationer

- **Requests** och **HTTPX** används för HTTP-kommunikation med externa API:er och tjänster [35, 36].

Datavalidering och struktur

- **Pydantic** och **Annotated-types** möjliggör typvalidering och strukturerad hantering av data, särskilt vid utbyte av JSON-data mellan komponenter [37, 38].

Miljöhantering och produktion

- **Python-dotenv** hanterar miljövariabler som API-nycklar och inställningar [39].
- **Gunicorn** och **Whitenoise** används i produktionsmiljö för att köra Django-applikationen (**Gunicorn**) och servera statiska filer (**Whitenoise**) [40, 41].

PDF- och textbehandling

- **PyPDF2** och **Fitz(PyMuPDF)** används för att extrahera text från PDF-filer, vilket möjliggör indexering av dokument [42, 43].
- **NLTK** och **Regex** används för enklare textförbehandling, exempelvis tokenisering och mönstermatchning [44, 45].

Mallhantering och säker HTML-rendering

- **Jinja2** och **MarkupSafe** används för mallrendering i Django och för att förhindra säkerhetsrisker vid HTML-generering [46, 47].

Utvecklingshjälpmedel

- **TQDM** används för att visa förloppsindikatorer under långa beräkningar och processer, vilket underlättar felsökning och övervakning av pipelines [48].

Dessa bibliotek bidrar till ett robust, skalbart och lättutvecklat system som fungerar väl både lokalt och i molnmiljöer.

4

Genomförande

Detta kapitel beskriver tillvägagångssättet som användes för att bygga projektet.

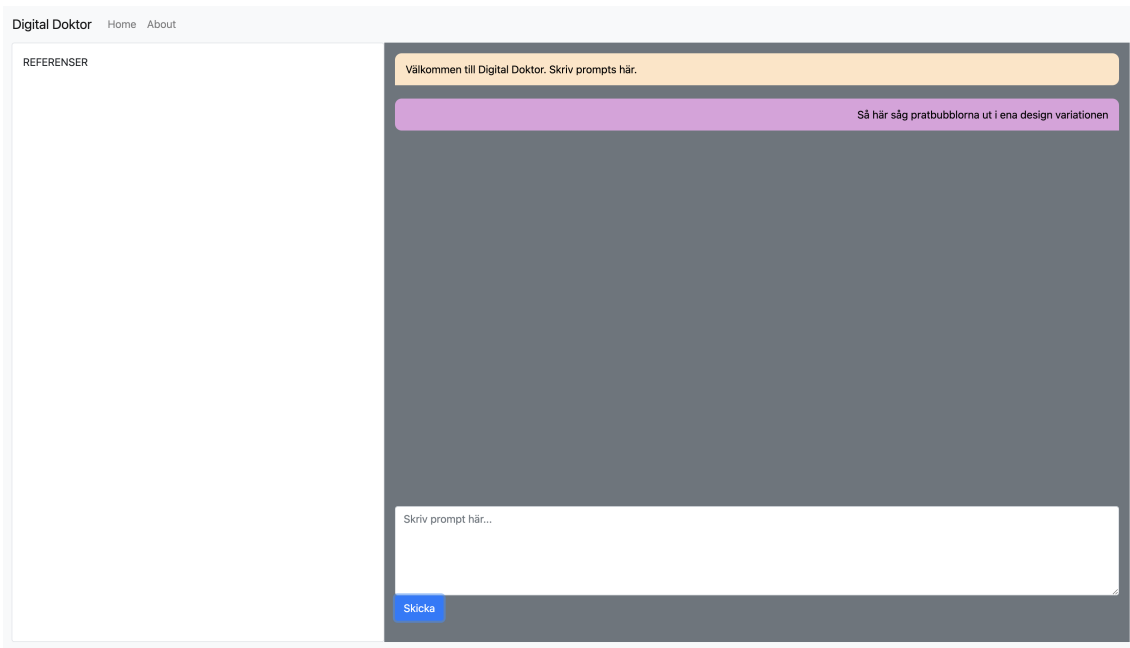
4.1 Utvecklingsfas av webbapplikation

I denna fas utvecklas basfunktionerna för att uppnå en applikation, samla in och bearbeta data, samt formulera frågor till LLM.

4.1.1 Design

Det första steget i projektet vara att bygga en webbapplikation. Målet med webbapplikationen var att användaren ska kunna skriva en fråga riktad till LLM med RAG, och sedan bli presenterad svaret på frågan med referenser till texter. Den ska vara enkel att navigera med ett simpelt gränssnitt. Det gjordes två separata design lösningar på uppgiften, sedan valdes den som ansågs bäst lämpat för arbetet.

Det första design alternativet var baserat på att konversationen skulle likna en typisk chat konversation med pratbubblor. Referenserna skulle även presenteras i en separat spalt på vänster sida om chatten. Se figur 4.1. Fördelen med denna designen var att det går att enkelt se temporär historik. Historiken är temporär då svaren och frågorna ej sparas utan endast finns kvar på webbsidan tills det att den uppdateras. Nackdelen blir att det kan upplevas av användaren som att svaren kommer baseras på tidigare frågor och svar. Det är inte fallet och därför anses denna design bidra med förvirring, vilket inte ska förekomma i en applikation som har målet att ge tydliga råd. Valet att inte ha en chatt som baserar svar på framtida frågor på historiken, baseras på risken att det ställs frågor menade för två olika typer av patienter i samma chatt. LLM ska inte bli förvirrad och tro att exempelvis en fråga om hur borrelia ska behandlas för en vuxen ska påverka nästa fråga om hur stor dos av ett läkemedel ett barn med astma ska få.



Figur 4.1: Figuren visar designalternativet för webbapplikation som var baserat på pratbubblor.

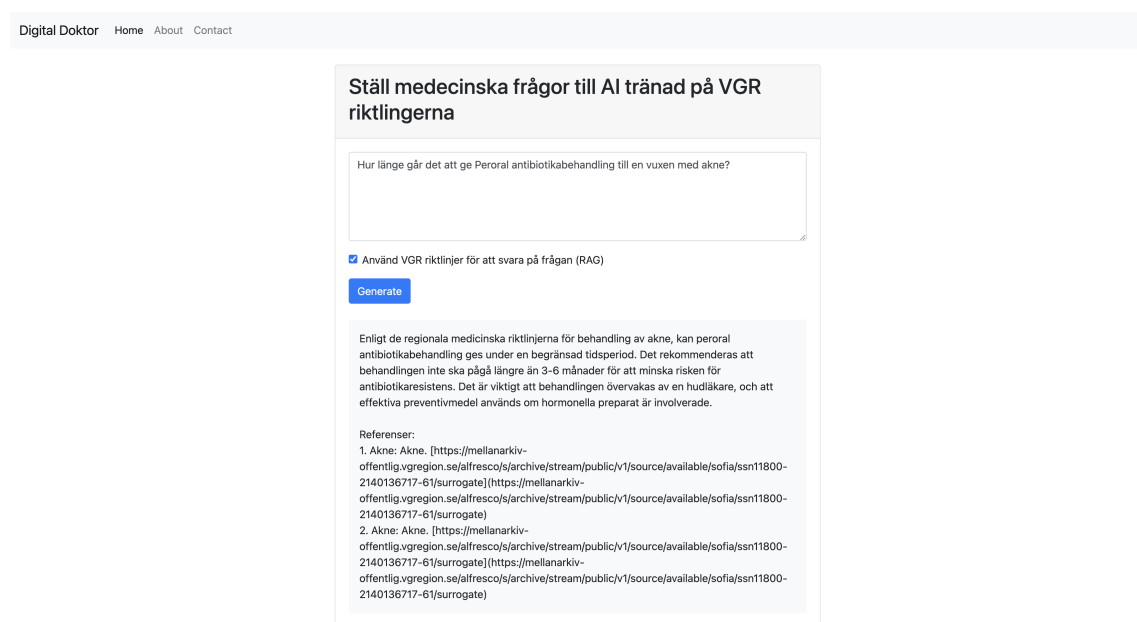
Det andra alternativet var en simplare version där det inte visas temporär historik utan endast den fråga som ställts senast, och vad svaret på den blev. Det är inte pratbubblor utan en textbox där frågan ställs och en under där svaret presenteras. Källorna kommer att presenteras längst ned i svarets textbox. Se figur 4.2. Denna design ansågs vara enklare att använda, då man minimerar förvirringen. Användaren ser endast frågan som ställts och svaret på den frågan. Det blev denna variation som webbapplikationen sedan baserades på. Detta val av design är fördelaktigt för att analysera svaren från LLM:en i utvecklingsfasen.

4.1.2 Bygga webbapplikationen

Webbapplikationen byggdes med hjälp av Django ramverket. För varje sida i webbapplikationen skapades en URL för åtkomst till denna sida. Varje URL kallar på en funktion i views när sidan ska visas. Views innehåller alltså en funktion per sida i webbapplikationen. Denna funktion hanterar visningen av sidan genom att kalla på HTML sidan, men även vilka backend funktioner som kallas på vid olika event på webbsidan. HTML sidorna skapades med hjälp av Bootstrap, och sparades i undermappen templates vilket är standarden inom Django.

För skapandet av hemsidan användes HTML och JavaScript. Hemsidan baseras på ett HTML formulär med metoden POST. För att skicka frågan som skrivs in på hemsidan vidare till backend funktionerna skapades ett event på “Generera” knappen, se figur 4.2. Eventet skapar en variabel av typ FormData som innehåller informationen från formuläret som backend funktionerna behöver ha. Dessa är frågan som ställts samt om användaren valt att använda RAG eller inte, anledningen till funktionen att inte ha RAG på hela tiden är att säkerställa att calls till LLM:en funkar och kommer vara en temporär funktion under utvecklingsfasen. För att ge backend tillgång

till denna data görs ett FETCH anrop i funktionen. FETCH funktion är uppbyggd av metod, body och headers. Metoden är POST, body är formdatan och headers innehåller information om att det är en XMLHttpRequest och det CSRF token som finns i formuläret. Funktionen förväntar sig sedan ett svar från backend som innehåller svaret från LLM:en. Detta svar publiceras sen i svars textboxen på hemsidan. När det uppstår fel i backend så kommer inget svar att ges utan endast "error". I början av utvecklingen innan kontakten till backend lades till fanns problemet att innehållet i frågan direkt försvann när användaren triggade knapp eventet. Eftersom det är ett formulär så är det en del av standard beteendet att texten försvinner när formuläret skickas, detta förhindras i vårt projekt, och frågan står kvar tills dess att den raderas av användaren.



Figur 4.2: Figuren visar designalternativet för webbapplikation som var baserat på svar i textbox.

4.1.3 Hämta ned riktlinjer från Västra Götalandsregionens webbsida

För att utföra RAG behövdes data som LLM kan basera svaren på. Datan i detta projekt fanns som PDF:er på Västra Götalands Regionens webbsida. Eftersom det är en stor mängd PDF:er skulle det vara omständigt att ladda ned varje dokument för hand och därför gjordes en webbskrapning med hjälp av Python biblioteket BeautifulSoup. Resultatet sparades i en csv fil som innehöll titeln på PDF:erna samt tillhörande URL:er. Dessa kunde sedan användas för att hämta ned PDF:erna från hemsidan så att dessa kunde bearbetas för att uppnå våra mål. Webbskrapningen görs bara en gång och vid uppdaterade riktlinjer måste därför en ny webbskrapning utföras.

4.1.4 Bearbetning av PDF:er

En viktig del av arbetet var att bearbeta data, som var VGR riktlinjerna i form av PDF:er. För varje PDF extraherades texten med hjälp av fitz biblioteket som används för att hantera PDF:er i Python. Fitz öppnar PDF:en som matchar URL:en den får som parameter. Strukturen bevaras och därför extraheras texten sida för sida med hjälp av den inbyggda funktionen “get_text”. Texten från PDF:en behövde sedan bearbetas vidare vilket gjordes genom att omvandla all text till små bokstäver, ta bort alla special tecken samt att filtrera bort vanliga svenska ord som finns i texterna men som inte ger utökad förståelse om riktlinjer. Dessa ord kallas stoppord och några exempel på dessa är “på” och “som”. För att filtrera bort stopporden användes nltk biblioteket stopwords, som innehåller de vanligaste stopporden i flera språk, i detta projekt applicerades endast de svenska orden. Utöver nlks stoppord applicerades även stoppord som var relevanta för den specifika kontexten. Texten sparades ned som .txt filer, vars namn var titeln för pdf dokumentet. Det skapades även en JSON fil kallad metadata som innehöll originaltiteln, URL till PDF:en hos VGR:s hemsida samt filsökvägen till både PDF:en och den bearbetade textfilen för alla PDF:er som separata objekt.

4.1.5 Första versionen av inbäddningar & sökande efter relevanta textbitar i datan

För att göra det möjligt för LLM:en att hitta relevant data till frågorna som ställs, måste det göras inbäddningar av all data. Den första versionen av inbäddningar på datan använde Sentence Transformers biblioteket i Python. Modellen som valdes för sentencetransformningen var “all-MiniLM-L6-v2” som hanterar inbäddningar av meningar och små paragrafer. Texten delades upp i delar om 512 bitar med ett överlapp på 50. Överlapp används för att det inte ska försvinna kontext om en mening eller paragraf skulle delas upp i på ostrategiska ställen. För varje del sparades titeln och URL:en samt bitens id. Det var på dessa bitar inbäddningen utfördes med hjälp av tidigare nämnd modell. Faiss biblioteket användes sedan för att producera likheterna mellan de olika inbäddade bitarna, med hjälp av den inbyggda funktionen “IndexFlatL2”. Resultatet blir distansen mellan alla bitar med data, vilket ger likheten mellan dem. Denna information används sedan för att hitta de relevanta bitarna text till frågorna som ställs till vårt program, och sparas i variabeln “index”. Efter inbäddningarna och indexering över likheter mellan de olika bitarna data var klar, blev nästa del i projektet att hämta data baserat på frågan som ställts. Detta gjordes genom att först göra en inbäddning av frågan med samma modell som använts för inbäddningarna av data. Sedan användes Faiss inbyggda funktion “search” för att hitta de bitar data som bäst liknar frågan. Visuellt går det att föreställas sig att det görs genom att hitta de databitar som hamnar närmast frågan i kluster av olika databitar. Parametern som bestämde högst antalet databitar som kan returneras var “top_k”, vilket sattes till tre, då det ansågs en rimlig mängd databitar som kan refereras. De databitar som matchar bäst till frågan sparas ned i resultatet som innehåller texten, titeln, URL:en, databitens id och hur relevant databiten är till frågan. Hur relevant databiten är beräknas genom att ta ett minus distansen

mellan databiten och frågan från “search” funktionen efter att den normaliserats. Detta ger ett tal mellan noll och ett som beskriver hur stor likheten är till frågan.

Notera att vi inte använder någon klustringsmetod (såsom k-means eller HNSW) i vår implementation, utan tillämpar en ’brute-force’ sökning med FAISS IndexFlatL2, vilket jämför frågans vektor med alla inbäddade databitar direkt. Detta ger hög precision men kan vara mindre effektivt för mycket stora datamängder då det tar lång tid. Men i projektet prioriterades precision, därmed valdes IndexFlatL2.

Metod	Indexeringstid	Söktid	Exakt	Skalbarhet
Brute Force (IndexFlatL2)	$\mathcal{O}(1)$	$\mathcal{O}(N \cdot D)$	Ja	Dålig
K-means Clustering (IndexIVFFlat)	$\mathcal{O}(N \cdot D \cdot I)$	$\mathcal{O}(\frac{N}{K} \cdot D)$	Nej	Bra

Tabell 4.1: Jämförelse mellan brute force och k-means klustringsbaserad vektorsökning. N är antalet vektorer, D är vektordimension, I är antal iterationer i k-means, och K är antal kluster [49].

4.1.6 Formulera uppdaterad fråga med kontext till LLM

Nästa steg i utvecklingen av projektet blev att formulera den uppdaterade frågan som ska ställas till LLM:en baserat på frågan som ställts till vårt program, och de relevanta databitarna för den frågan. Detta ska göras i naturligt språk. Först formuleras kontext, vilket är information om databitarna som skickas vidare till LLM:en.

```
context = "Relevant information from medical guidelines:\n\n"
"
for i, result in enumerate(results):
    context += f"Document {i+1}: {result['metadata']['title']}\n"
    context += f"Source: {result['metadata']['url']}\n"
    context += f"Excerpt: {result['text']}\n\n"
```

Hela frågan till LLM:en blir som följande.

```
enhanced_prompt = f"" Please answer the following medical
question based on the provided information.

{context}

Question: {prompt}

Please provide a concise answer and include references to
the specific medical guidelines you used.
""
```

4.1.7 Testar RAG med llma3

Innan projektet kopplades upp till LLM:en GPT-4o-mini som används i projektet gjordes tester på llma3 för att se om informationen som skickades gav svar. Llama3 användes då det är en fri att använda, och det gick att testa att köra applikationen utan att hantera API nycklar från Azure. Tyvärr hade programmet det svårt att hitta relevant data när frågor ställdes. I vissa fall refererade svaren till riktlinjer, men i andra fall hävdade programmet att det ej fanns riktlinjer på ämnet, trots att det fanns med i PDF:erna. Det var inte önskat beteende men slutsatsen att kontext och referenser hanterades på önsakt vis av LLM även om kontexten ej innehöll allt som önskats. Det beslutades att nästa steg blev att sätta upp användning av den faktiska LLM:en som skulle användas av programmet, GPT-4o-mini.

4.2 Distrubitionsfas

I distrubitionsfasen hanteras kopplingen till LLM:ets API samt publicering av web-applikationen via Azure.

4.2.1 Koppla LLM API till projektet

GPT-4o-mini kördes via Azure portalen. Först startades en ny openAI resurs som sedan behövs sättas i drift. Härifrån fås API nyckeln till LLM:en, slutpunkten för resursen i azure och driftnamnet av modellen. Denna information lades in i nycklarna som programmet kan använda. Första steget i att kalla på LLM:en var att skapa en klient som innehåller den relevanta datan för konsumtion av OpenAIs API, vilket är vilken version av API:et som önskas användas, vart webbapplikationen finns i Azure, samt nyckeln till API:et. För att kalla på LLM:en skapades en chatt med ett meddelande som bestod av frågan som ställs till LLM:en samt rollen på användaren som är "user". Förfrågan till API:et innehöll utöver chattmeddelandet även information om vilken GPT modell som används samt de olika parametrarna som justeras för generering av svar. Temperaturen avgör hur kreativa svaren ska vara, och i detta fall då det inte önskas genereras en berättelse utan fakta baserat på data valdes en låg temperatur. Top_p avgör hur fokuserat det genererade svaret ska vara, genom att filtrera svaren baserat på hur stor andel av de troliga tokens som kan användas i generering av svaret. Eftersom det önskas säkra och fokuserade svar i denna applikationen kommer alltså ett lågt värde på top_p användas. Max_tokens avgör hur långt svaret som längst kan bli. GPT-4o-mini kan som mest generera 16,384 tokens [20], så 4096 blir ungefär en fjärdedel. Att ha mindre tokens än vad som är möjligt som maximum baserades på att svaren inte skulle bli kortfattade nog annars.

```
def get_azure_openai_response(prompt):  
  
    client = AzureOpenAI(  
        api_version=AZURE_OPENAI_API_VERSION,  
        azure_endpoint=AZURE_OPENAI_ENDPOINT,  
        api_key=AZURE_OPENAI_API_KEY,  
    )
```

```
try:
    response = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": prompt,
            }
        ],
        max_tokens=4096,
        temperature=0.2,
        top_p=0.5,
        model=AZURE_DEPLOYMENT_NAME
    )
    return response.choices[0].message.content

except Exception as e:
    print(f"Error calling Azure OpenAI: {str(e)}")
    return f"Error generating response: {str(e)}"
```

4.2.2 Publicera webbapplikation

Projektet använde GitHub för att underlätta samarbetet och göra versions hantering möjligt. GitHub projektet kunde sedan kopplas till webbapplikations resursen som skapades i Azure. Genom att publicera GitHub projektet hos Azure gick det att publicera applikationen till internet där det går att nyttjas av alla så länge applikationen är igång på Azure. För att Azure skulle hitta vart projektet ska köras lades det till en startup.sh fil som innehöll sökvägen till wsgi.py som hanterar distribueringen och startar applikationen. För att möjliggöra detta lades startup.sh in som filsökväg i Azure inställningarna för start av programmet.

4.3 Vidareutvecklingsfas

När basfunktioner fanns och arbetet hanterade API:er, var det dags att förbättra programmet, så att det fungerade enligt målen.

4.3.1 Testa första modellen

Trots att applikationen nu använde en starkare LLM så var resultaten de samma som i tidigare test med llma3. Den hittade vissa referenser på specifika frågor, exempelvis var den bra på information om diabetes men kunde ej hitta information om akne trots att det fanns en hel PDF på ämnet. Det var inte förvånande då problemet antogs vara kontexten som skickades till LLM:en. Där med blev nästa steg i utvecklingen att undersöka förbättringsområden i sökandet för relevant data.

4.3.2 Undersöka förbättringsområden

Det första som undersöktes var om mängden tabeller som PDF:erna innehöll kunde påverka inbäddningarna negativt då formateringen inte behölls i de bearbetade textfilerna. För att undersöka detta användes biblioteken `unstructured` och `pdfplumber`. `Unstructured` biblioteket kan användas för hantering av PDF:er i Python, och `pdf partitioner` användes för att dela upp data i varje PDF två grupper, tabeller eller text. Då skulle formateringen av tabellerna behållas. Tyvärr var det för tungt för datorn det kördes på och därför byttes `unstructured` ut mot `pdfplumber`. `pdfplumber` kunde även det extrahera tabeller och text från PDF:erna. Informationen sparades i JSONobjekt som innehöll sidnummer, texten, samt om det var tabell eller inte. Om det var en tabell erhöles även tabellnummer. Att dela upp data i tabeller och ren text hjälpte tyvärr inte resultaten, då RAG fortfarande inte hittade bra textbitar från riktlinjerna att skicka, vilket resulterade i att undersöka möjligheter att göra om inbäddningar och byte av modell.

4.3.3 Andra versionen av inbäddningar & sökande efter relevanta textbitar i datan

För att förbättra inbäddningar så används OpenAIs egna inbäddningsmodell “text-embedding-ada-002” istället för den tidigare sentencetransformern “MiniLM-L6-v2”. “text-embedding-ada-002” hanteras via klient anrop till openAIs tjänst genom Azure. Alla dessa inbäddningar sparas i en pickle fil för serialisering. Alla databitar innehåller fortfarande samma information som tidigare och det går därmed att hitta tillhörande titel samt URL för all text. Faiss används fortfarande på samma vis för att jämföra de olika databitarna som inbäddats på nytt vis. Frågan som ställs till programmet, som det ska hittas relevanta databitar till, hanteras också genom klient anrop till den nya inbäddningsmodellen. Sedan används Faiss inbyggda funktion `search` som tidigare och hittar de `top_k` svaren, alltså de `top_k` databitar som bäst liknar frågan. Denna inbäddning fungerade betydligt bättre och vid test levererade den relevanta svar till frågor som ställdes, där det fanns riktlinjer.

4.4 Testning & justerings fas

När projektet ansågs fungera rent tekniskt övergick projektet till testnings och justerings fasen. Det skedde tester angående innehåll och även uppleved användarvänlighet samt tydligheten av svaren.

4.4.1 Initiala tester och justeringar

För att testa programmet letades rimliga frågor i några av PDF:erna. Sedan ställdes frågan och det utvärderades om svaret matchade texten som lästs, samt om programmet refererade till rimliga texter. Då vi som utvecklare inte har medicinsk kunskap var dessa frågor mest menade till att utvärdera om programmet tekniskt fungerade som förväntat. Ett fåtal frågor ställdes även av en läkare som innehar relevant kunskap för att utvärdera svaren. Baserat på dessa svar framgick det att

modellen inte gav alla möjliga lösningar på vissa frågor, och det var därmed rimligt att skicka fler databitar till LLM:en. Det valdes att öka till fem stycken ifrån tre, då detta gav tillfredsställande resultat på de frågor som ställts i den första rundan av mindre tester.

Vid testningen var ett önskemål att referenser skulle hanteras bättre, då LLM:en kunde göra att de uppfattades "röriga". Det löstes genom att ge det utökade prompten tydligare direktiv om att det skulle hanteras med "[i]", som är IEEE standard. Referenslistan hanterades istället separat från det publicerade svaret. Detta gjordes genom att hämta alla nummer som fanns inuti hakparenteser, och sedan i en egen textbox publicera dessa nummer med tillhörande titel och URL. Dock kunde flera databitar ge referenser till samma dokument, men de kommer ändå att ha olika referens nummer. Dessa "dubletter" skulle bli en och samma referens. Utöver det så kan referenserna börja på exempelvis två om inte första databiten användes i svaret detta behövde också hanteras, så att de börjar på ett som konventioner ofta kräver, samt att det blir mer användarvänligt. Därför sparades istället positionen av hakparenteserna i svaret ned tillsammans med URL:en, och ett nummer som den delgavs ner i en lista. Programmet går igenom alla element i listan och adderar den första instansen av varje URL tillsammans med en siffra den delges, siffran börjar på ett för första hakparentesen och varje ny unik URL får siffran efter. Detta för att referenserna alltid ska starta på ett och att numret ska öka ju senare i svaret som det återfinns. Sedan med hjälp av placeringar av hakparenteserna som hittats tidigare av programmet går det att skriva över värdet i dem med referensnumret som tilldelats aktuell URL. Därefter hanteras dubletterna som uppkommit genom att skapa ett mönster som tar bort dessa från svaret. Detta mönstret hittar alla hakparenteser på följd som innehåller samma siffra. Sedan används informationen om vilka URL:er som använts samt vilken siffra de delgivits för att konstruera referens listan, som innehåller tilldelad siffra, titel och URL. Referensnummerna skulle vara länkar till riktlinjen och där med skapades en funktion "makeReferencesClickable()" som hittade mönster i referenserna och gjorde om innehållet i referenslistan till länkar med URL delen av mönstret som sökväg, och den andra informationen fick vara kvar som ren text.

Det var även ett önskemål att programmet skulle svara på frågor baserat på tidigare frågor och svar, vilket inte hade ansetts vara en del av projektet tidigare. Detta gjorde att första webbsidans design med "pratbubblor" återigen blev aktuell, och backend kopplades ihop med denna. Historik löstes genom att hämta sessions baserad historik som fås av förfrågan till event på hemsidan. History är en lista som innehåller frågor till programmet samt svar från LLM:en utan referenser i text. Att referenserna inte skickas med är för att inte förvirra LLM:en när historik skickas med till den. Följande kodexempel visar hur fråga och svar sparas till historiken.

```
history.append(  
    {"role": "user", "content": original_prompt}  
)  
history.append(  
    {"role": "system", "content":  
        llm_responce_without_ref}  
)
```

Det första som sker när backend nås av frågan är att programmet hämtar sessions historiken om sådan finns, annars blir det en tom lista. Finns det historik så kallar programmet på en funktion som har frågan till programmet samt historiken som parametrar. Ett klientanrop till LLM:en utförs där den bestämmer vilka element i historiken som är relevant för frågan som ställdes till programmet senast. LLM:en svarar i form av en lista med de index från historik listan som var relevanta och ska behållas. Alla andra tas bort i historiken, då de inte ska påverka framtida svar. Den nya historiken som endast består av relevanta element läggs till i frågan som ställdes till programmet. Likt tidigare hittas relevanta databitar i riktlinjer baserat på denna fråga, men nu innehåller den mer kontext om vad som sagts tidigare. Det betyder att att om ämnet hanterade en behandling för ett hjärtproblem och användaren skriver exempelvis “Blir man trött av det?” Så kommer databitar om den hjärtbehandlingen och biverkningar vara de som programmet ger till LLM:en och inte data om trötthet i allmänhet. Frågan till LLM:en har inte ändrats utan det är endast kontexten som nu är anpassad efter historiken. Att en LLM bestämmer vilka historik element som är relevanta krävs då endast fem databitar från riktlinjerna hanteras av LLM:en och därför måste dessa vara de som är mest relevanta till senaste ställda frågan, och inte blanda in information från tidigare chattar om annat ämne.

4.4.2 Omfattande testning

För att utvärdera modellen krävdes mer omfattande tester i en andra omgång. Här skapades ett antal medicinska frågor av en ST-läkare, som sedan utvärderade svaren på frågorna. Efter detta utvärderades det vilka justeringar som skulle vara rimliga för att förbättra modellen. De olika alternativen för utvärdering av svaren var:

- Bra rekommendation
- Helhet bra - saknas något
- Helhet bra - finns fel formuleringar eller onödig text
- Gick inte att följa råd
- Hittar inte relevant data
- Fel

Viktigt att poängtera är att användning av alternativet “Hittar inte relevant data”, kan bero både på brister av programmets sökande av relevant information, men även att ämnet inte återspeglas i riktlinjerna som programmet har tillgång till. Det kan där med bli svårt att dra tydliga slutsatser baserat på denna utan att gå igenom all data. Det är fortfarande viktigt att alternativet finns då det går att dra slutsatser om hur stor användning regionen skulle få av ett program tränat på deras offentliga riktlinjer.

5

Resultat

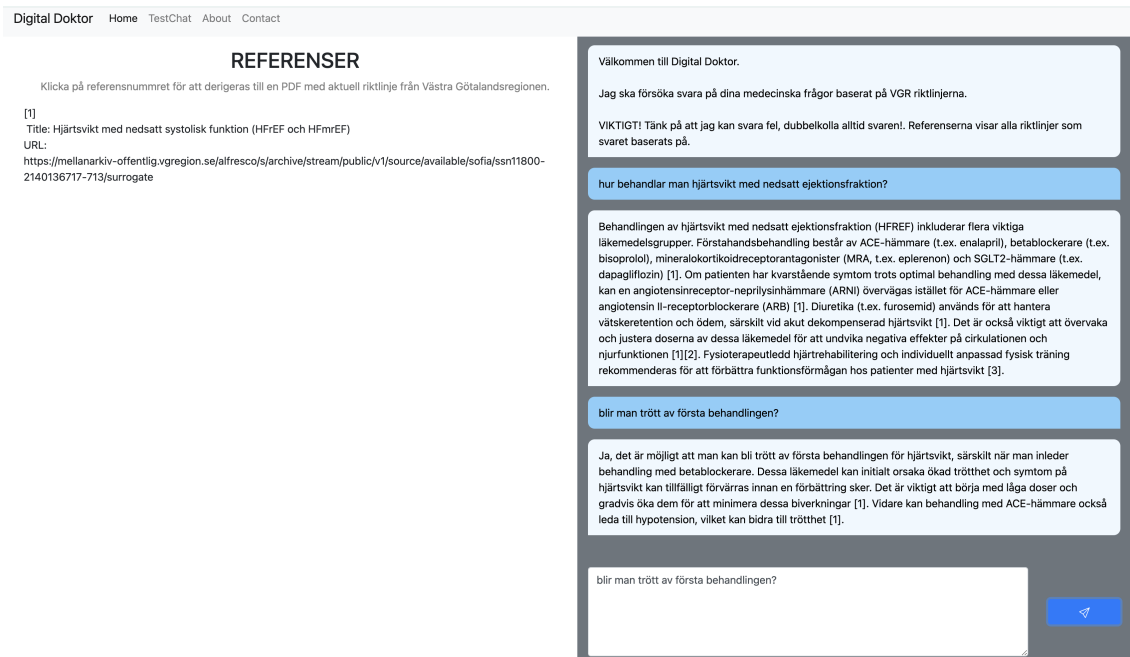
I denna delen av rapporten presenteras resultaten av testningen som utfördes samt presentation av den slutgiltiga produkten.

5.1 Utseende

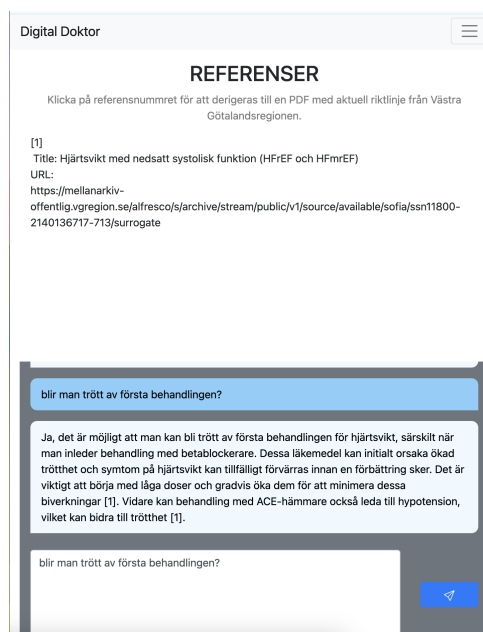
Webbsidan är användarvänlig och går att använda på flertalet olika enheter samt webbläsare. Det enda som användaren behöver göra är att skriva en fråga till programmet. Svar och referenser genereras automatiskt. Webbapplikationens layout är uppdelad i två delar, en för presentation av aktuella referenser och en för chatten med programmet. Se figur 5.1. En referens presenteras med referensens nummer i en hakparentes, samt titel och URL som tillhör dokumentet som det referenserar. Nummret är en länk som det går att klicka på för att komma till URL:en. Referenserna presenteras i kronologisk ordning baserat på chattsvaret. Endast det senaste svarets källor presenteras i referenslistan. Chattedelen av applikationen är uppdelad i två. En fast del i form av textbox där frågor kan ställas och knappen för att skicka frågan, samt en del med skrollfunktion där ställda frågor och svar presenteras i form av pratbubblor. Varje gång ett meddelande skrivs läggs det till i fönstret och det skrollar automatiskt längs ned för att se senaste meddelande först. Pratbubblorna presenteras i olika nyanser av ljusblå för att det ska vara enkelt att separera frågor från svar, men även för att färgerna är behagliga att läsa text på. Pratbubblorna har även designade på så vis att de har en rak kant på vänster sida för programmets svar och en rak kant på höger sida för användarens fråga. Det är för att efterlikna pratbubblor. Vid uppdatering av sidan eller om sidan stängs ned kommer alla meddelanden och referenser försvinna då dessa inte sparas.

Designen på webbapplikationen är responsiv och går att använda på så väl datorer som mobiler. När bredden minskar så byter applikationen layout från radbaserat till kolumnbaserat. Se figur 5.2. Utöver positionen av delarna har menyn även blivit en så kallad hamburgermeny. Vid klick på menyn så rullas hela designen ned och får en ny övre del som består av meny som presenteras vertikalt. Vid val av menyalternativ, alternativt vid klick på menyknappen så stängs menyn och hela designen återgår till original läget.

5. Resultat



Figur 5.1: Figuren visar den slutgiltiga designen av webbapplikationen.



Figur 5.2: Figuren visar hur den responsiva designen fungerar i den slutgiltiga versionen av webbapplikationen, där layouten ändras från rad- till kolumnbaserad när skärmbredden minskar.

5.2 Svaren från programmet

När användaren av programmet ställer en fråga, kommer ett svar som baseras på VGR riktlinjer att presenteras inom några få sekunder. Finns det ingen informa-

tion som anses relevant för frågan i riktlinjerna som programmet har tillgång till, kommer svaret att informera om detta, och inga referenser presenteras. Det kommer ändå att genereras ett svar som presenteras för användaren. Det blir därför viktigt för användaren att vara uppmärksam på att svaret endast kommer ifrån en LLM och inte är baserad på VGR riktlinjer. Programmet är menat för medicinska frågor, och frågor om andra ämnen än sjukvård kommer därmed inte att gå att mätas med svaren från de vanliga LLM chattbottarna idag. Svaren är kortfattade och ger endast nödvändig information baserat på frågan. Referenserna i meddelandet presenterar för varje bit av relevant information, vilket riktlinjes dokument informationen baseras på. Källorna presenteras med relevant referensnummer vid sidan om senaste svaret i en egen kolumn, dessa visar namn på artikeln och URL, detta är för att användaren ska ha möjlighet att kontrollera svaren, samt öka tilliten till svaren från programmet. Referenserna i svaret hanteras kronologiskt och dokumentet som används först kommer att få nummer ett. Även om flertalet dokument ansågs relevanta vid hämtning av dokumenten i programmet så kommer endast de som bidrog till programmets svar att presenteras för användaren.

5.3 Historik

Meddelanden som skickas till programmet är inte isolerade utan den tar tidigare chattar i åtanke när den svarar på frågor. Det krävs inte att användaren själv renser historiken när nytt ämne ska diskuteras, utan programmet bedömer om ett nytt ämne startats, och vilka tidigare meddelande som anses relevanta. De meddelanden som ej ansågs relevanta baserat på senaste frågan blir borttagna ur den interna historiken i programmet, för att endast relevanta databitar ifrån dokumenten ska hämtas och skickas vidare till LLM:en som genererar ett svar på frågan. Vid ställd följdfråga, minimerar det risken för att programmet svarar på ett tidigare fall som ej är relevant till nuvarande frågeställning. Referenserna är endast anpassade efter det senaste meddelandet och därför är nummer i gamla meddelanden inte längre relevanta för användaren. Det kan medföra förvirring för användaren att äldre meddelanden ej hänvisar till källor längre.

5.4 Kvalitativ analys av svar

Den kvalitativa testningen gjordes av ST-läkare som även jämförde svaren med ChatGPT. Först presenteras ett urplock av data som testades under ett möte med en ST-läkare. Dessa presenteras i tabell 5.1 nedan. Vid anblick går det att se att den klara majoriteten av svaren har bedömts ge bra rekommendationer, som vårdpersonal kan följa.

Tabell 5.1: Utvärdering av svar från programmet

Fråga	Kommentar
Vilka patienter med aortastenosis ska åtgärdas?	Bra rekommendation
Hur handlägger man patienter med en asymptomatisk aortastenosis	Bra rekommendation
Vilka ska opereras respektive vilka ska få perkutan klaff TAVI"	Bra rekommendation
Vilka ska göra EKG?	Helhet bra – saknas något
Vilka patienter med diabetes borde få en CGM?	Bra rekommendation
Vilka diabetespatienter bör inte få en CGM?	Bra rekommendation
Hur behandlas ångest?	Bra rekommendation
Ska man ge antibiotika till profylax till en extraktion?	Bra rekommendation
Vilka patienter bör få en ICD-behandling	Hittar inte relevant data

Tabell 5.2 nedan presenterar resultatet av jämförelsen med Chat-GPT. Frågorna presenteras i första kolumnen, sedan presenteras bedömningen av Chat-GPTs svar och sist bedömningen av programmet. Majoriteten av svaren klassades som bra rekommendationer i vårt program men det framgår att de kunde vara mer detaljrika.

Tabell 5.2: Utvärdering av svar från ChatGPT och DigitalDoktor

Fråga	ChatGPT-svar	DigitalDoktor-svar
Fråga 1: När ska man behandla ångest med SSRI?	Bra rekommendation	Bra rekommendation
Fråga 2: Hur diagnostiseras egentlig depression?	Bra rekommendation	Gick inte att följa råd – kriterierna för egentlig depression listades inte
Fråga 3: Vad kännetecknar en psykos?	Bra rekommendation	Bra rekommendation
Fråga 4: Vilken behandling ges vid PTSD?	Bra rekommendation	Helhet bra - saknas något
Fråga 5: Hur bedömer man suicidrisk?	Bra rekommendation	Gick inte att följa råd
Fråga 6: Hur behandlar man hjärtsvikt med nedsatt ejektionsfraktion?	Bra rekommendation	Helhet bra – saknas något
Fråga 7: När ska patienter med aortastenosis åtgärdas?	Bra rekommendation	Helhet bra – saknas något
Fråga 8: Hur handläggs förmaksflimmer?	Bra rekommendation	Helhet bra – saknas något
Fråga 9: Vad är målblodtryck vid hypertoni?	Bra rekommendation	Bra rekommendation – men ChatGPT har mer detaljer
Fråga 10: Akut handläggning vid STEMI?	Bra rekommendation	Bra rekommendation – men ChatGPT har mer detaljer
Fråga 11: Ska man ge antibiotika i profylax vid tandextraktion?	Bra rekommendation	Bra rekommendation

Fråga	ChatGPT-svar	DigitalDoktor-svar
Fråga 12: Hur förebyggs DVT postoperativt?	Bra rekommendation	Bra rekommendation
Fråga 13: Vilka vacciner rekommenderas till äldre?	Bra rekommendation	Bra rekommendation – men ChatGPT har mer detaljer
Fråga 14: När ges endokarditprofylax?	Bra rekommendation	Bra rekommendation – men ChatGPT har mer detaljer
Fråga 15: Antibiotikaprofylax vid kirurgi – när?	Bra rekommendation	Bra rekommendation – men ChatGPT har mer detaljer

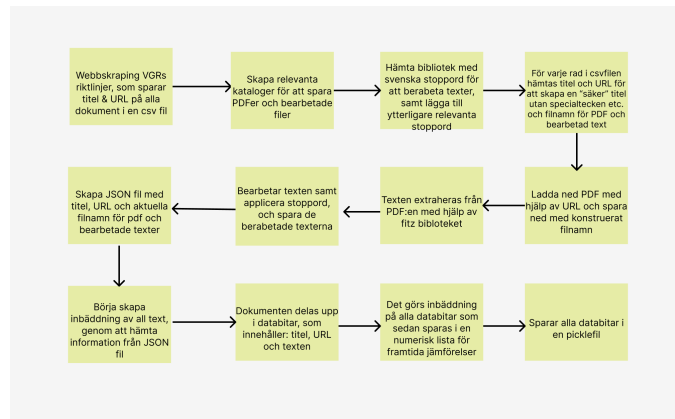
Tabell 5.3 visar en sammanställning av frekvensen av de olika alternativen. Även om det hade varit önskvärt med mer data får vi dra slutsatser baserat på det som finns. Två tredjedelar av svaren klassificerades som bra rekommendationer, även om det kunde varit önskvärt med mer detaljerade svar i flera av fallen. Helheten var bra men det saknades något i cirka en femtedel av svaren. Vid två tillfällen gick det inte att följa råden som presenterades. Vid ett tillfälle så hittades inte relevant data. Men som tidigare nämnt blir det svårt att dra slutsats om felet ligger i den tekniska implementationen eller om det är avsaknad av riktlinjer som påverkar resultatet. Vi kan se att Chat-GPT var bättre i många fall i jämförelse med vårt program.

Tabell 5.3: Sammanställd bedömning av programmet

Alternativ	Antal
Bra rekommendation	16
Helhet bra - saknas något	5
Helhet bra - finns fel formuleringar eller onödig text	0
Gick inte att följa råd	2
Hittar inte relevant data	1
Fel	0

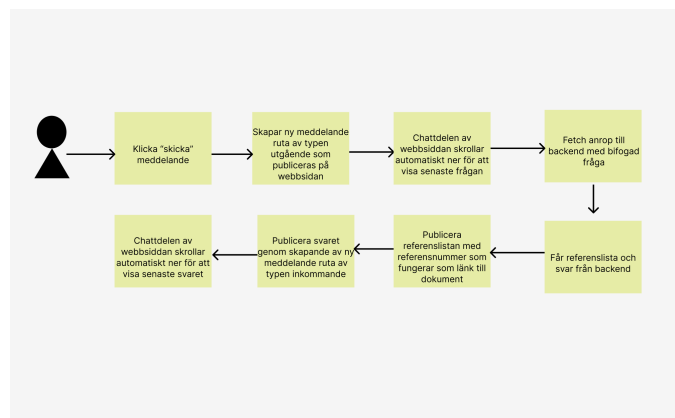
5.5 Systemkonstruktion

Nedan presenteras en övergripande bild över funktionerna som måste utföras innan körning av programmet kan påbörjas. Dessa görs endast en gång, samt om riktlinjerna skulle uppdateras så att ny information måste hämtas ned.

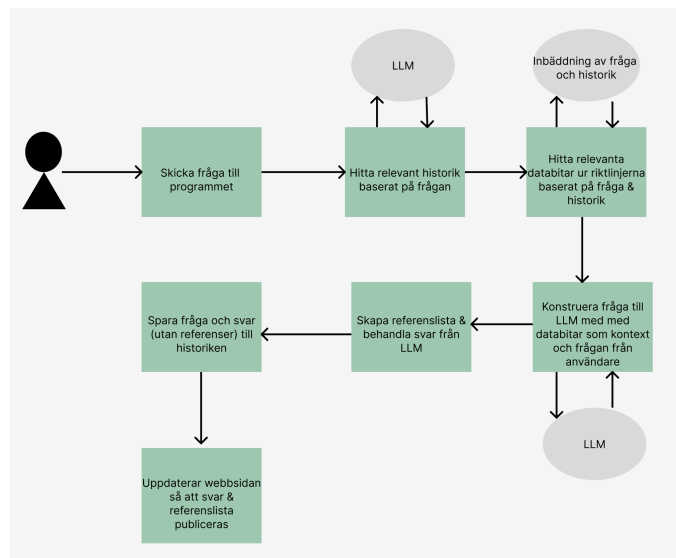


Figur 5.3: Figuren visar övergripande schema över hur programmet hämtar ned riktlinjer och bearbetar dessa för användning vid körning.

Nedan presenteras en enkel övergripande bild av vad som händer under körning, från dess att en fråga ställs tills att svar finns publicerade till användare. Funktionerna presenteras i kronologisk ordning utifrån hur de exekveras, snarare än att delas upp efter teknisk implementation eller kodstruktur. Se figur 5.4 för frontend fokuserad kronologisk överblick. Se figur 5.5 för backend fokuserad kronologisk överblick.



Figur 5.4: Figuren visar övergripande schema över hur programmet fungerar under körning, med fokus på frontend-funktionalitet.



Figur 5.5: Figuren visar hur programmet fungerar under körning, med en övergripande översikt av funktionaliteten i backend.

6

Slutsats

Projektet har visat på en potential för att använda AI-lösningar som stöd för att hantera vanliga frågor som idag ställs till läkare. Resultaten är baserade på regionala riktlinjer vilket ökar tilliten till svaren. Därmed anses det att projektets syfte uppfyllts.

Målet med applikationen uppnåddes då arbetet har resulterat i en prototyp som under utveckling och i mindre tester har bedömts ge användaren till majoriteten pålitliga svar med tydliga referenser till riktlinjer. Dock visade testerna på att det var mindre utförliga svar än vad som gavs av Chat-GPT och det kan därmed vara rimligt att ändra på prompten till LLM:en för att få bättre formulerade svar. Programmet förhåller sig till innehållet i VGRs riktlinjer och det gjordes bedömning av medicinskt sakkunnig att de kan vara för sparsmakade för att få ännu mer detaljerade svar. Det slogs fasta i den kvalitativa testningen att svaren var korrekta enligt informationen som fanns i riktlinjerna, men att det där inte fanns tillräckligt mycket information för ett utförligt svar i vissa fall. Därmed kan det krävas mer utveckling för att uppnå en färdig produkt för bredare användning. Men det har blivit en prototyp som uppfyller målen. Alla tester har visat på korrekta referenser när riktlinjer hittats av programmet. Webbapplikationen var uppbyggd i Python och använder ett AI API, samt att svaren anpassas genom RAG, vilket var krav i målet. Projektet hölls även inom de uppsatta avgränsningarna. Sammanfattningsvis går det att konstatera att prototypen visar på en potential för liknande projekt inom RAG att agera som stöd för att bidra till minskade mänskliga fel inom vården.

7

Diskussion

I detta kapitel tas det upp förslag på potentiell vidareutveckling samt en diskussion kring relevanta ekologiska, etiska och samhällsliga aspekter.

7.1 Utvecklingsmöjligheter

För att prototypen skulle kunna användas i praktiken hade en bättre UX för webbsidan varit relevant. Programmet är enkelt att använda och fungerar på olika typer av plattformar, men det finns utrymme för förbättringar. Programmet visar källreferenser med klickbara länkar som tar användaren till en ny sida, rimligen hade dessa istället öppnat dokumentet i samma fönster för att undvika att lämna webbsidan. Referenserna i meddelanden och i nummer i referenslistan skulle kunna sticka ut mer genom exempelvis bakgrundsfärg som gör att de syns tydligare och att det då ska upplevas mer intuitivt att använda programmet. Vidare skulle även hantering av svaren från programmet, kunnat hanteras som klickbara objekt, för att återfå tillgång till vilka referenser som meddelandet hänvisar till genom att dessa återfinns i referenslistan när meddelandet är markerat.

Projektet använder sig av LLM:en GPT-4o-mini, men en undersökning av LLM:er anpassade till medicinsk data är ett relevant steg i vidareutvecklingen för att se om de fungerar bättre på detta ämne. Det finns bibliotek för stopwords som är specifikt inriktade mot medicinska ord, som skulle kunna vara relevanta i detta sammanhang, då programmet i nuläget endast använder bibliotek med vanliga svenska ord som inte är anpassade för vården. Majoriteten av dessa bibliotek är dock fokuserade på engelska, och har betydligt mindre svenska ord. Detta programs data är endast baserat på offentlig information från regionen, men det finns annan data som programmet kan utökas med.

Eftersom LLM:en inte ska hantera obegränsat med tokens, blir det viktigt att sätta en gräns på antal ord som användaren får använda per meddelande. Vilket inte gjorts i vårt program men som blir en nödvändig förbättring. Det blir även viktigt att mängden meddelanden som skickas till LLM för beslut om vilka meddelanden som är relevanta, sätter en gräns på hur många meddelanden den får behålla. Det krävs dock utvärdering om vad en rimlig mängd historik objekt blir. Programmet i nuläget sparar endast de viktigast och det har fungerat bra under utveckling och testning, men för en användbar produkt måste det vara utsatt specifikt. Det är för att LLM:en återigen inte ska hantera för många tokens även när det kommer till att utvärdera meddelanden. Att hantera många tokens skulle bli dyrt och onödigt i många fall.

7.1.1 Ekologiska aspekter

LLM:er som GPT-4 har en tydlig miljöpåverkan, både vid träning och användning. Det handlar bland annat om utsläpp av koldioxid [50]. I detta projekt användes en mindre modell, GPT-4o-mini, vilket innebär lägre energiförbrukning per fråga. Eftersom officiella siffror främst finns för GPT-4 används dessa som referens. Vi utgår från att vår modell ligger i ungefär samma storleksordning, åtminstone som uppskattning.

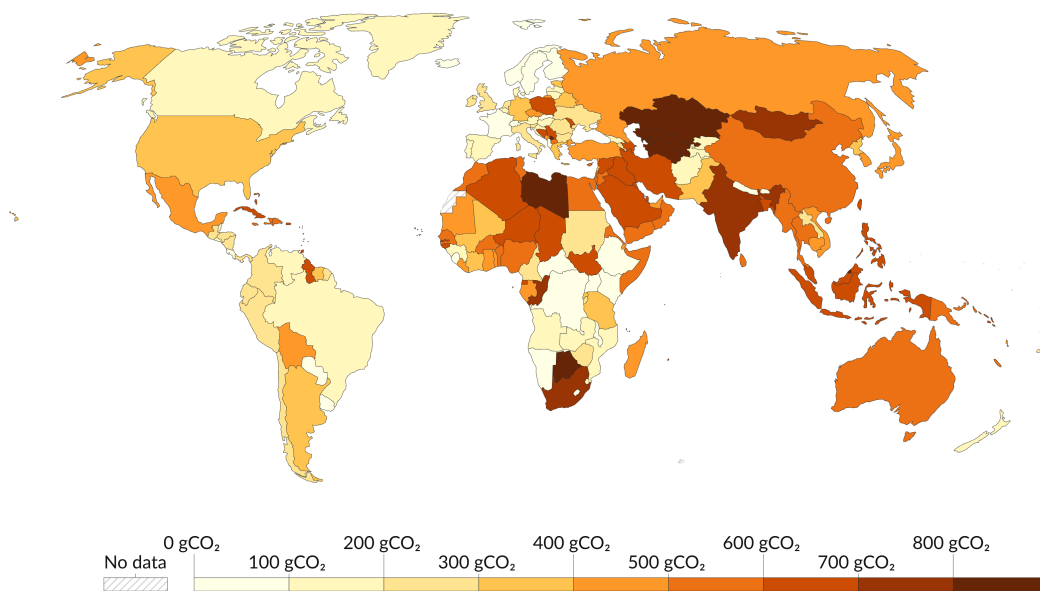
Träningen av GPT-4 krävde mellan 51 och 62 GWh, vilket motsvarar ungefär 0,05 procent av Sveriges årliga elförbrukning [51] [52]. Det är mycket energi, men inte orimligt med tanke på det stora antalet användare världen över. Utsläppen uppgår till mellan 1 245 och 1 500 ton koldioxid från träning, ungefär 0,5 procent av Sveriges elrelaterade utsläpp [1].

Även den löpande användningen påverkar klimatet. En enda fråga till en språkmodell drar mellan 0,3 och 40 wattimmar, beroende på hur komplex frågan är. I Sverige, där elen är relativt ren (ungefär 36 gram CO₂ per kWh), ger det utsläpp på mellan 0,01 och 1,4 gram CO₂ per fråga. I USA, med smutsigare el (cirka 384 gram CO₂ per kWh), kan siffran bli upp till 15 gram.

Carbon intensity of electricity generation, 2024

Our World
in Data

Carbon intensity is measured in grams of carbon dioxide-equivalents emitted per kilowatt-hour of electricity generated.



Data source: Ember (2025); Energy Institute - Statistical Review of World Energy (2024)

OurWorldinData.org/energy | CC BY

Figur 7.1: Skillnad i koldioxidutsläpp per producerad kilowattimme el mellan olika länder [1].

Klimatpåverkan från LLM:er varierar alltså kraftigt beroende på var de används. En rapport uppskattar att den globala användningen av GPT-4 kan leda till utsläpp på upp till 30 000 ton CO₂ per år, ungefär 10 procent av Sveriges elrelaterade utsläpp [53]. I det här fallet handlar det dock om ett verktyg riktat till vårdpersonal, och användningen förväntas bli betydligt mindre.

Att hoppa över 50 000 frågor till en språkmodell sparar lika mycket utsläpp som att byta ut alla glödlampor mot LED i ett hushåll [54]. Samtidigt varierar påverkan per fråga kraftigt. En enkel prompt kan dra lika mycket el som fem minuters laptopanvändning (1 Wh), medan en komplex fråga kan motsvara att ha på en mikrovågsugn i 30 sekunder (9 Wh) eller mer.

Sammanfattningsvis är energiförbrukningen från stora språkmodeller inte obetydlig, men heller inte extrem i jämförelse med annan digital eller vardaglig elförbrukning. Miljöansvar handlar om att använda tekniken på ett effektivt och rättvist sätt. Det innebär att fundera över om modellens elförbrukning är rimlig med tanke på vad den används till. I vissa fall kan det vara bättre att utveckla lösningar där öppna modeller körs lokalt på egen hårdvara med el från förnybara källor, särskilt om externa tjänster använder mer resurser än vad som motsvarar deras faktiska värde [55].

7.1.2 Etiska & samhällsmässiga aspekter

LLM:er väcker flera etiska och samhällsrelaterade frågor, särskilt vid användning inom sjukvård. En central utmaning är att modellerna inte alltid ger samma svar på identiska frågor, eftersom de är sannolikhetsbaserade snarare än deterministiska. Detta kan skapa osäkerhet eller misstro, särskilt i sammanhang där konsekvens och förutsägbarhet är avgörande. Till skillnad från läkare, som kan motivera sina slutsatser logiskt och transparent, saknar modellen förståelse för sina egna svar. Därför bör vissa svar vara fördefinierade och granskade av medicinsk expertis. Om modellen används som ett avancerat sökverktyg, likt en sofistikerad **Ctrl+F**-funktion, bör ansvaret för innehållet ligga på utvecklarna. Används den däremot för att identifiera samband eller generera hypoteser, måste vårdpersonalen ta ansvar för att värdera och tolka resultaten.

Vem får ansvaret om det blir fel? Det kan bli ett etiskt dilemma. Mänskliga misstag är en del av vårdverksamheten och hanteras genom rutiner, dubbelgranskning och ansvarsfördelning. När ett AI-verktyg gör fel är det däremot oklart vem som bär ansvaret. I praktiken ligger det fortfarande på vårdpersonalen, men detta förutsätter att AI:n ses som ett beslutsunderlag. Här finns en risk för ansvarsglidning, där användaren tror på modellens svar utan att förstå hur de uppstått [2].

Om modellen visar sig effektiv ökar risken för ett beroende, inte tekniskt utan kognitivt. Användare kan förlora vana att självständigt tolka riktlinjer eller dra slutsatser. Liknande mönster har setts inom andra områden, exempelvis hur GPS:en berättar vart vi ska åka idag, så vi tolkar inte kartor själva i samma grad. Tekniken

ersätter inte bara manuellt arbete, utan förändrar också vilka färdigheter som upprätthålls. En till liknelse kan göras med tvättmaskinen. Få saknar kunskapen att tvätta för hand, men det spelar ingen roll så länge maskinen fungerar. Skillnaden är att AI-system ofta tillhandahålls av kommersiella aktörer med osäker finansiering. Exempelvis drivs GPT-4 av ett bolag som ännu inte uppnått lönsamhet. Det väcker frågor om långsiktig tillgång. Därför är det viktigt att utveckla hållbara lösningar och ha backup-planer, både för mjukvaran och för vårdpersonalens kompetens, ifall en tjänst skulle försvinna eller bli föråldrad.

AI-baserade verktyg kan bidra till färre fel genom snabb och konsekvent informationssökning. Men det förutsätter att användaren förstår verktygets begränsningar. Därför krävs utbildning, inte bara i användning utan även i grundläggande förståelse för hur modeller fungerar. Detta bör inkluderas i vårdens utbildningar, om program likt denna aktivt skulle användas i vården.

Det etiska ansvaret omfattar även produktionen. Utvinning av sällsynta jordartsmetaller, som krävs för hårdvaran, sker ofta under tveksamma arbetsförhållanden [56]. Produktionen och driften av dessa system är dessutom elintensiv, vilket gör elens ursprung avgörande ur hållbarhetssynpunkt. Hela värdekedjan bör granskas, från elproduktion och komponenttillverkning till återvinning. Frågor om datasuveränitet och geopolitik blir också alltmer relevanta [57]. Exempelvis kan kontroll över serverinfrastruktur eller tillgång till råmaterial påverka både tillgänglighet och säkerhet.

Den nuvarande prototypen hanterar ingen patientdata, vilket minskar risken för brott mot dataskyddsförordningen (GDPR). Om modellen i framtiden används i kliniska miljöer krävs strikt efterlevnad av dataskyddsregler. Det måste även vara tydligt att det medicinska ansvaret alltid vilar på vårdpersonalen.

Sammanfattningsvis finns betydande potential av AI. Vid korrekt användning, kan tekniken minska felbehandlingar och effektivisera informationssökning. Dock måste en produkt som aktivt används i vården ge bättre svar än denna prototyp. Men balansen mellan nytta och risk kräver tydliga riktlinjer, kontinuerlig utvärdering och kompetensutveckling.

Litteraturförteckning

- [1] Our World in Data. Carbon intensity of electricity generation. <https://ourworldindata.org/grapher/carbon-intensity-electricity>, 2025. Hämtad 2025-06-16.
- [2] Donaldson MS Kohn LT, Corrigan JM. To err is human: Building a safer health system, 2000. Hämtad 2025-04-01. URL: <https://www.ncbi.nlm.nih.gov/books/NBK225179/>.
- [3] Västra Götalandsregionen. Medicinska riktlinjer och rutiner, 2024-10-18. Hämtad 2025-04-2. URL: <https://www.vgregion.se/halsa-och-varld/varldgivarwebben/varldriktlinjer/medicinska-och-varldadministrativa-riktlinjer/styrande-dokument-inom-halso--och-sjukvard/>.
- [4] Subir Gupta, Pushan Kumar Dutta, Soumojit Roy, Aparna Bhaduri, Kalyan Maji, Abhik Choudhary, and Asmita Sarkar. Evaluating waterfall vs. agile models in software development for efficiency and adaptability. *IGI Global*, page 142 – 148, 2024. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85196468520&doi=10.4018%2f979-8-3693-3318-1.ch008&partnerID=40&md5=4cf3b9a332298dcc44c58e8ea3d105af>, doi: 10.4018/979-8-3693-3318-1.ch008.
- [5] USA Robert F. Rose Alexandria, VA. Software development activity cycles | collaborative development, continuous testing and user acceptance, 2022. Hämtad 2025-04-2. URL: <https://link.springer.com/book/10.1007/978-1-4842-8239-7>.
- [6] Django Software Foundation. Why django?, 2025. Hämtad 2025-04-2. URL: <https://www.djangoproject.com/start/overview/>.
- [7] Daniel Rubio. *Beginning Django: Web Application Development and Deployment with Python*. Apress Berkeley, CA, 1 edition, 2017. doi:<https://doi-org.proxy.lib.chalmers.se/10.1007/978-1-4842-2787-9>.
- [8] David Hemmendinger. Html, 2025-03-18. Hämtad 2025-04-4. URL: <https://www.britannica.com/technology/HTML>.
- [9] MA Kte'pi, Bill. Cascading style sheets (css), 2025. Hämtad 2025-04-4. URL: <https://research.ebsco.com/c/lu54te/viewer/html/7eqddr5ppb>.
- [10] Holger M. Kienle. It's about time to take javascript (more) seriously. *IEEE Software*, 27(3):60 – 62, 2010. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-77951687486&doi=10.1109%2fMS.2010.76&partnerID=40&md5=e28a0de865cc86b1b8a29d37b39d4e2f>, doi:10.1109/MS.2010.76.
- [11] Tutorialspoint. What is javascript?, nd. Hämtad 2025-04-29. URL: https://www.tutorialspoint.com/javascript/javascript_overview.htm.

- [12] Bootstrap. Get started with bootstrap, nd. Hämtad 2025-04-29. URL: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>.
- [13] Python Software Foundation. General python faq, 2025. Hämtad 2025-04-2. URL: <https://docs.python.org/3/faq/general.html#general-python-faq>.
- [14] Ke Li. The influence of python on logical thinking and analysis of teaching countermeasures. *Journal of Physics: Conference Series*, 1744(4):042164, feb 2021. URL: <https://dx.doi.org/10.1088/1742-6596/1744/4/042164>, doi: 10.1088/1742-6596/1744/4/042164.
- [15] MA Kte'pi, Bill. Så här fungerar generativ ai och llm:er, 2025. Hämtad 2025-04-4. URL: <https://learn.microsoft.com/sv-se/dotnet/ai/conceptual/how-genai-and-llms-work>.
- [16] ollama. Run large language models locally. <https://ollama.com/>, 2025. hämtad 2025-05-27.
- [17] Sentence-transformers/all-minilm-l6-v2, nd. Hämtad 2025-05-20. URL: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>.
- [18] Microsoft. Vad är stora språkmodeller (llm:er)?, 2024-11-24. Hämtad 2025-04-29. URL: <https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-are-large-language-models-llms>.
- [19] OpenAI. The fastest and most powerful platform for building ai products, nd. Hämtad 2025-04-29. URL: <https://openai.com/api/>.
- [20] OpenAI. Gpt-4o mini, nd. Hämtad 2025-04-29. URL: <https://platform.openai.com/docs/models/compare?model=gpt-4o-mini>.
- [21] Microsoft. Inbäddningar i .net, 2024-12-21. Hämtad 2025-04-29. URL: <https://learn.microsoft.com/sv-se/dotnet/ai/conceptual/embeddings>.
- [22] Rick Merritt. What is retrieval-augmented generation, aka rag?, 31-01-2025. Hämtad 2025-04-29. URL: <https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/>.
- [23] Vad är azure, nd. Hämtad 2025-05-20. URL: <https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-is-azure/>.
- [24] Why github, nd. Hämtad 2025-05-20. URL: <https://github.com/why-github>.
- [25] Beautiful soup documentation, nd. Hämtad 2025-05-5. URL: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>.
- [26] psycopg2. Postgresql database adapter for python. <https://www.psycopg.org/>, 2024. hämtad 2025-05-27.
- [27] dj-database url. Use database urls in django. <https://github.com/jacobian/dj-database-url>, 2024. hämtad 2025-05-27.
- [28] pandas. Powerful python data analysis toolkit. <https://pandas.pydata.org/>, 2024. hämtad 2025-05-27.
- [29] numpy. Numpy. <https://numpy.org/>, 2024. hämtad 2025-05-27.
- [30] scipy. Scipy. <https://scipy.org/>, 2024. hämtad 2025-05-27.
- [31] scikit learn. Machine learning in python. <https://scikit-learn.org/>, 2024. hämtad 2025-05-27.
- [32] faiss. Facebook ai similarity search. <https://github.com/facebookresearch/faiss>, 2024. hämtad 2025-05-27.

-
- [33] joblib. Running python functions as pipeline jobs. <https://joblib.readthedocs.io/>, 2024. hämtad 2025-05-27.
- [34] threadpoolctl. Python helpers to limit thread usage. <https://github.com/joblib/threadpoolctl>, 2024. hämtad 2025-05-27.
- [35] requests. Http for humans. <https://docs.python-requests.org/>, 2024. hämtad 2025-05-27.
- [36] httpx. A next-generation http client for python. <https://www.python-httpx.org/>, 2024. hämtad 2025-05-27.
- [37] pydantic. Data validation using python type hints. <https://docs.pydantic.dev/>, 2024. hämtad 2025-05-27.
- [38] annotated types. Annotated types for python. <https://pypi.org/project/annotated-types/>, 2024. hämtad 2025-05-27.
- [39] python-dotenv. Python dotenv support. <https://pypi.org/project/python-dotenv/>, 2025. hämtad 2025-05-27.
- [40] gunicorn. Wsgi http server for unix. <https://docs.djangoproject.com/en/5.2/howto/deployment/wsgi/gunicorn/>, 2025. hämtad 2025-05-27.
- [41] whitenoise. Static file serving for wsgi apps. <http://whitenoise.evans.io/>, 2024. hämtad 2025-05-27.
- [42] PyPDF2. Read and write pdfs with python. <https://pypi.org/project/PyPDF2/>, 2024. hämtad 2025-05-27.
- [43] PyMuPDF. Python bindings for mupdf (fitz). <https://pymupdf.readthedocs.io/>, 2024. hämtad 2025-05-27.
- [44] nltk. Natural language toolkit. <https://www.nltk.org/>, 2024. hämtad 2025-05-27.
- [45] regex. Alternative regular expression module. <https://pypi.org/project/regex/>, 2024. hämtad 2025-05-27.
- [46] jinja2. Modern and designer-friendly templating. <https://jinja.palletsprojects.com/>, 2024. hämtad 2025-05-27.
- [47] markupsafe. Safely add untrusted strings to html/xml. <https://pypi.org/project/MarkupSafe/>, 2024. hämtad 2025-05-27.
- [48] tqdm. Fast, extensible progress bar for python. <https://tqdm.github.io/>, 2024. hämtad 2025-05-27.
- [49] Faiss indexes, nd. Hämtad 2025-05-20. URL: <https://github.com/facebookresearch/faiss/wiki/Faiss-indexes>.
- [50] Gábor Szarmes, Peter och Éllő. Sustainability of large ai models: Balancing environmental and social impact with technology and regulations. *Chemical Engineering Transactions*, 107:103 – 108, 2023. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85183654840&doi=10.3303%2fCET23107018&partnerID=40&md5=4723b4fd37a4d28d7a79d58f2142b182>, doi:10.3303/CET23107018.
- [51] Kasper Groes Albin Ludvigsen. The carbon footprint of gpt-4. <https://medium.com/data-science/the-carbon-footprint-of-gpt-4-d6c676eb21ae>, 2023. hämtad 2025-06-16.
- [52] Svenska kraftnät. Elstatistik. <https://www.svk.se/om-kraftsystemet/kraftsystemdata/elstatistik/>, 2024. hämtad 2025-06-16.

- [53] Josh You. How much energy does chatgpt use?, 2025. Accessed: 2025-06-17. URL: <https://epoch.ai/gradient-updates/how-much-energy-does-chatgpt-use>.
- [54] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís-Miquel Mun-guia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emis-sions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.
- [55] 2018.
- [56] Amnesty International. This is what we die for: Human rights abuses in the democratic republic of congo power the global trade in cobalt. Technical report, Amnesty International, 2016. URL: <https://www.amnesty.org/en/documents/afr62/3183/2016/en/>.
- [57] Franz Jessen Sebastian Contin Trillo-Figueroa. Will trump’s ‘po-lar pivot’ spark a superpower clash? *Asia Times*, 2025. Ac-cessed: 2025-06-17. URL: <https://asiatimes.com/2025/01/will-trumps-polar-pivot-spark-a-superpower-clash/>.

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige

www.chalmers.se



CHALMERS