Verbetermogelijkheden RAG

Inleiding

Bij het innovatieteam van SSC-ICT zijn wij bezig met retrieval augmented generation, oftewel RAG. Het basisidee van RAG is dat je (interne) documenten kan koppelen aan een groot taalmodel (een LLM). Grote taalmodellen zijn modellen zoals Chat GPT, Claude, LLama en Mistral die tekst kunnen genereren op basis van de input van de gebruiker.

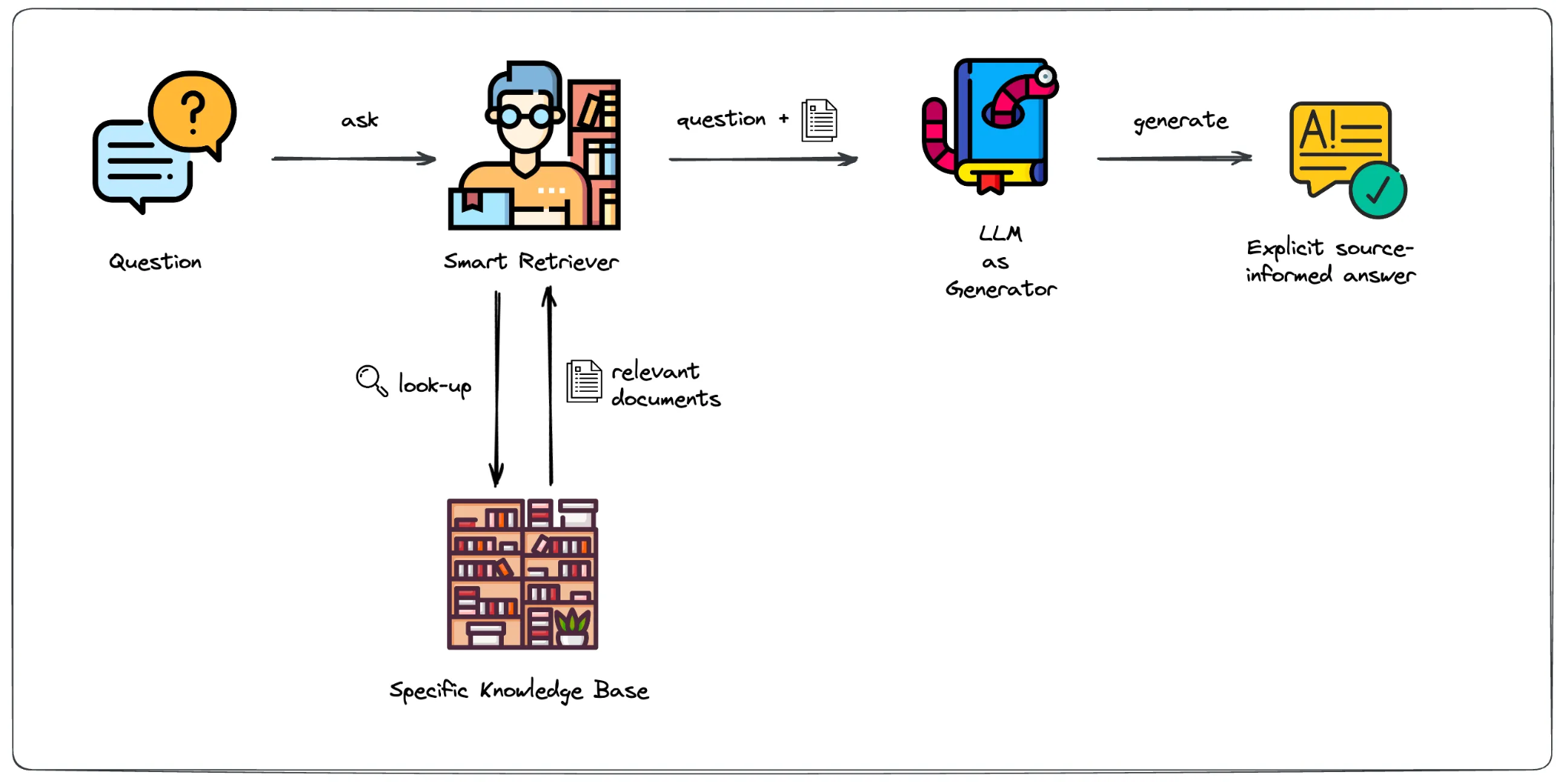
[Onze RAG-applicatie](https://github.com/SSC-ICT-Innovatie/LearningLion) werkt (grofweg) als volgt:

*Voor de gebruiker een vraag stelt:*

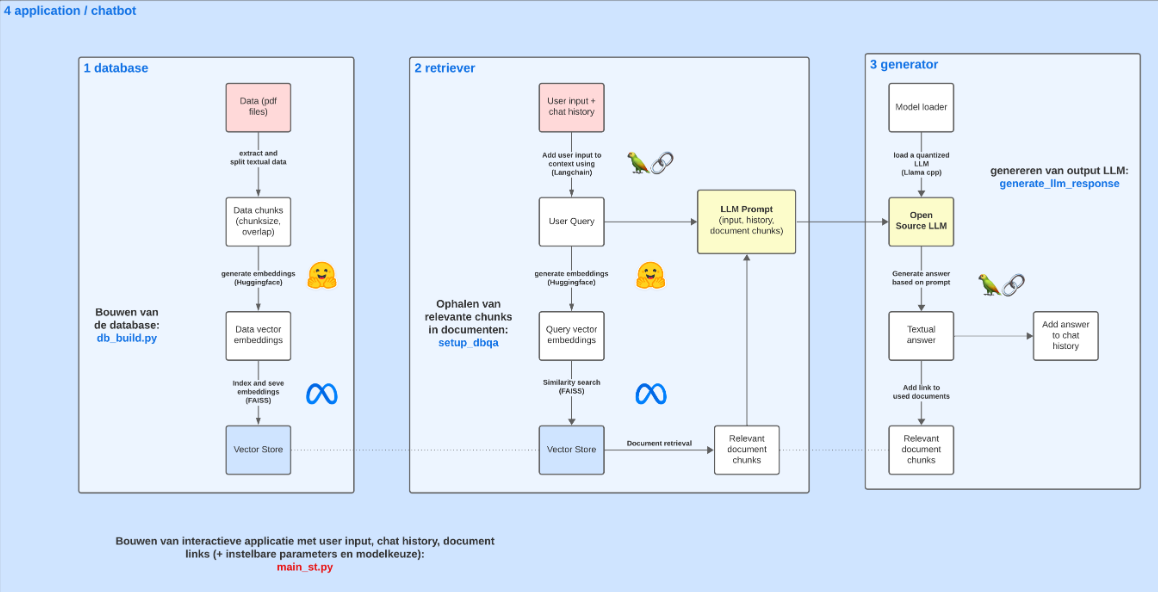
1. Je laadt de documenten die je wilt bevragen in, deze documenten worden opgeknipt in stukjes (chunks) van een grofweg vaste, van tevoren bepaalde lengte. Bijvoorbeeld chunks van 1024 tokens (circa 700 woorden) met een overlap van 256 tokens (circa 175 woorden). Dit noemen we chunking
2. Deze chunks worden omgezet in een numerieke weergave (ookwel vector) door een embedding model, dit model is erop getraind om 2 stukken tekst met een vergelijkbare **betekenis** een vergelijkbare numerieke weergave te geven. Dit noemen we het embedded of vectorizer. Deze numerieke weergaven worden opgeslagen, de plek waar deze opgeslagen worden noemen we de vector store.

*Wanneer de gebruiker een vraag stelt:*

1. De vraag van de gebruiker wordt door hetzelfde embedding model omgezet tot numerieke weergave
2. De numerieke weergave van de vraag wordt vergeleken met de numerieke weergave van alle chunks via een similarity search. Vervolgens wordt er berekend welke chunks de laagste afstand tot de vraag hebben. Gezien het embedding model erop getraind is om 2 stukken tekst met een vergelijkbare betekenis een vergelijkbare numerieke weergave te geven, zou dit tot de chunks moeten leiden die qua betekenis dicht bij de vraag liggen en dus relevant zijn voor de vraag.
3. Deze chunks worden toegevoegd aan de input van de gebruiker, en aan het LLM wordt gevraagd de vraag van de gebruiker te beantwoorden op basis van de gegeven chunks. Deze chunks waar het antwoord op gebaseerd wordt krijgt de gebruiker ook te zien.



*Eenvoudige visuele weergave*



*Uitgebreide visuele weergave met de libraries en applicaties die wij gebruikt hebben*

In ons [visiestuk](https://docs.google.com/document/d/1J6k0HdlY5NMrOtSI0UhMYBZSoIlaqGwMx-FXzhio3Vk/edit?usp=sharing) kan je lezen waarom wij voor RAG gekozen hebben en welke stappen wij denken dat er gezet moeten worden. Als voorbeeld kan je eraan denken dat het nuttig is wanneer je beleid maakt om makkelijk te kunnen vragen of er al eerder iets vergelijkbaars is geprobeerd en wat de evaluaties daarvan waren (in dit geval zou dus de database volstaan met beleidsevaluaties en rapportages).

Reden voor document

De manier waarop wij RAG vormgeven wordt ook wel [simpele RAG](https://medium.com/enterprise-rag/an-introduction-to-rag-and-simple-complex-rag-9c3aa9bd017b) genoemd, en werkt verre van perfect. Zo worden belangrijke stukken tekst niet altijd gevonden, of blijken er vragen te zijn waarvan het antwoord niet in een specifiek stukje tekst staat maar die desondanks op basis van de beschikbare documenten te beantwoorden zijn. Daarom is er de laatste tijd veel onderzoek gedaan naar tal van manieren om RAG te verbeteren.

In de rest van dit document zullen de verschillende mogelijke verbeteringen beschreven worden. Voor de lezer die visueel ingesteld is, is hier een filmpje waar een aantal van deze concepten besproken worden: [Building Production-Ready RAG Applications: Jerry Liu - YouTube](https://www.youtube.com/watch?v=TRjq7t2Ms5I). Daarnaast komt er veel terug in deze artikelen:

1. [Advanced RAG Techniques: an Illustrated Overview | by IVAN ILIN | Dec, 2023 | Towards AI](https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6)
2. [Technical Considerations for Complex RAG (Retrieval Augmented Generation) | by Chia Jeng Yang | Enterprise RAG | Dec, 2023 | Medium](https://medium.com/enterprise-rag/a-first-intro-to-complex-rag-retrieval-augmented-generation-a8624d70090f)
3. [A Guide on 12 Tuning Strategies for Production-Ready RAG Applications | by Leonie Monigatti | Dec, 2023 | Towards Data Science](https://towardsdatascience.com/a-guide-on-12-tuning-strategies-for-production-ready-rag-applications-7ca646833439)

Deze opties heb ik onderverdeeld in een tweetal opgave en categorie overig, feedback zeer welkom over wat er mist en wat wel/niet de moeite waard is.

Opgave 1: Hoe zorg je ervoor dat je de relevante stukken tekst vindt?

**Probleemstelling**

De ervaring leert dat de gebruikte similarity search verre van perfect is, heel vaak worden passages die cruciaal zijn om een correct en volledig antwoord op de vraag te geven niet gevonden. Wanneer deze niet gevonden worden, is het onmogelijk voor het taalmodel een hoog kwaliteit antwoord te geven, hoe goed het taalmodel ook is. Het basisprobleem zit hier vaak in de methodiek. In feite wordt er gezocht naar de stukken tekst die (qua betekenis) het meest lijken op de vraag, maar lijken op de vraag blijkt niet gelijk aan relevant zijn voor het antwoord.

**Opdracht**

Om toch de relevante stukken tekst te vinden is er een breed scala aan technieken ontwikkeld om de zoekfunctie te verbeteren. Sommige dingen zullen heel snel gaan, bij andere zullen er complicaties optreden die we nu nog niet kunnen overzien.

**Technieken**

Metadata opslaan en filteren

Een tijdje terug had ik alle verkiezingsprogramma’s van politieke partijen ingeladen als database, vervolgens vroeg ik het systeem wat D66 wilde doen tegen dakloosheid. In geen enkele chunk ging het zowel over D66 als over dakloosheid, immers in het verkiezingsprogramma van D66 was beleid rond dakloosheid in de ‘wij’ vorm beschreven. Naar wie ‘wij’ verwees was onduidelijk uit die chunk. Ondertussen ging het in het verkiezingsprogramma van de SP heel uitgebreid over dakloosheid, dus de embedding van dat stuk leek het meest op mijn vraag. Als antwoord kreeg ik wat SP aan dakloosheid wilde doen.

Dit probleem kan makkelijk verholpen worden als het systeem opslaat uit welk document welke chunk komt, en dat vervolgens meegenomen wordt in de retrieval. Daarom wordt vaak geadviseerd zogenaamde ‘meta-data’ mee op te slaan met iedere chunk. Metadata gaat om dingen als in welk document iets staat, op welke pagina, wie dat document geschreven heeft, wanneer het geschreven is. Voor veel vragen is het evident dat alleen chunks die bepaalde metadata hebben relevant zijn, mijn vraag over D66 plannen kan alleen beantwoord worden met tekst uit het D66 verkiezingsprogramma. Vragen over wat er in 2011 ergens over geschreven is alleen met stukken uit 2011. Het filteren op bepaalde metadata alvorens er chunks geselecteerd worden kan dus een goede toevoeging zijn.

Zie: [Advanced RAG: Optimizing Retrieval with Additional Context & MetaData using LlamaIndex🦙 | by Akash Mathur | Dec, 2023 | Medium](https://medium.com/@akash-mathur/advanced-rag-optimizing-retrieval-with-additional-context-metadata-using-llamaindex-aeaa32d7aa2f)

Samenvattingen opslaan en doorzoeken

Met metadata kan je filteren op de naam van een document, maar je zou ook op de inhoud van het document kunnen filteren. Dit kan je doen door van ieder document een samenvatting te maken (gelukkig zijn LLM’s daar vrij goed in dus hoef je dat niet zelf te doen) en voor je specifieke chunks zoekt eerst aan de hand van de samenvatting bepaald in welk document het waarschijnlijk staat, om vervolgens te kijken waar in dat document het hierover gaat.

Hybride zoeken

Veel zoekmachines kijken niet zo zeer naar de betekenis van een alinea maar zoeken op trefwoorden. Hoewel er veel voordelen zijn aan het zoeken naar betekenis, zijn er ook vragen waarbij het van cruciaal belang is dat specifieke trefwoorden terugkomen in de context. Als ik vraag tot hoe laat de Albert Heijn op zondag open is, is het cruciaal dat specifiek de woorden Albert Heijn en zondag in de context terugkomen (en niet woorden die kwa betekenis in de buurt liggen, zoals coop en zaterdag). In dit soort gevallen is de kans dat dit fout gaat kleiner als er op specifieke trefwoorden wordt gezocht dan op algemene betekenis. Echter als ik vraag hoe er in de media ergens over bericht wordt wil ik juist dat stukken die het over journalistiek, of de NOS hebben ook gevonden worden en niet specifiek alleen op het woord media gezocht wordt. Daarom blijkt in praktijk zogenaamd ‘hybride’ zoeken vaak het beste te werken, waarbij er afhankelijk van de vraag gekeken wordt of trefwoorden of betekenis relevanter zijn. In dit artikel: [Improving Retrieval Performance in RAG Pipelines with Hybrid Search | by Leonie Monigatti | Nov, 2023 | Towards Data Science](https://towardsdatascience.com/improving-retrieval-performance-in-rag-pipelines-with-hybrid-search-c75203c2f2f5) wordt er uitgebreid besproken hoe dit te implementeren is en wat de meerwaarde ervan is.

Hypothetische vragen of antwoorden embedden

Ik heb al meerdere keren meegemaakt dat ik een vraag stel en dan een stuk tekst terugkrijg waarin dezelfde vraag staat maar niet de tekst waarin het antwoord daarop staat. Dit is heel logisch want de vraag lijkt qua betekenis meer op de vraag dan op het antwoord. Soms lijkt het antwoord zelfs totaal niet op de vraag.

Om dit te ondervangen hebben mensen een zeer creatief idee bedacht wat in praktijk blijkt te werken[[1]](#footnote-0). Het komt erop neer dat je eerst de vraag laat beantwoorden door het taalmodel, waarbij je het expres een antwoord laat hallucineren. Dit antwoord klopt waarschijnlijk niet, maar qua zinsconstructie en grammatica lijkt het mogelijk wel meer op het daadwerkelijke antwoord dan de vraag, het heeft immers de vorm van een antwoord. Vervolgens ga je op dit antwoord de similarity search uitvoeren om tekst te zoeken die het daadwerkelijke antwoord op de vraag bevat. Dit [paper](http://boston.lti.cs.cmu.edu/luyug/HyDE/HyDE.pdf) is erover geschreven, en [hier](https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6) gaat deel 2.3 erover.

Dit kan je overigens ook aan de andere kant doen, bij ieder stukje tekst kan je ook nagaan welke vragen daarin beantwoord worden en die vragen opslaan om daar similarity search op uit te voeren met de daadwerkelijk gestelde vragen.

Klein zoeken, groot maken (hierarchisch zoeken)

Een ingewikkelde keuze bij RAG is hoe groot je de chunks maakt, maak je de chunks te groot dan is er een risico op een ‘lost in the middle’ situatie het feit dat er veel irrelevante tekst rond een relevante passage staat maakt het lastiger het goede stuk te vinden. Maak je de chunks te klein dan is de kans er dat cruciale context ontbreekt. Denk maar aan de situatie waar mijn vraag in een andere chunk stond dan het antwoord op die vraag.

Een manier om dit aan te pakken is door in stapjes te zoeken. Het idee is hier dat kleinere stukken tekst vaak meer informatie over relevantie bevatten, maar grotere stukken tekst de context die nodig is om een vraag volledig te beantwoorden. Het idee is dus dat we stapsgewijs zoeken, eerst wordt een kleine stuk tekst (bijvoorbeeld een zin) gevonden met een hoge similarity score om vervolgens niet alleen die zin maar ook de context eromheen mee te geven aan de LLM[[2]](#footnote-1).

Reranking

[Meerdere](https://blog.lancedb.com/simplest-method-to-improve-rag-pipeline-re-ranking-cf6eaec6d544) [artikelen](https://levelup.gitconnected.com/reranking-in-rag-pipline-with-flashrank-complely-free-ultralight-andd-fastest-reranker-a7380eab495e) suggereren dat er veel winst te behalen is door eerst een groot aantal relevante chunks te selecteren en vervolgens die weer te ranken om de meest relevante aan de LLM te geven. Het idee is dat de reranking accurater is omdat het meer relevante context over de relatie tussen de potentieel relevante tekst en de vraag kan meewegen, maar het minder efficiënt is dus het belangrijk is dat er al een voorselectie gemaakt is van potentieel relevante teksten. Bovendien krijg je uiteindelijk stukken tekst die door twee aparte modellen als relevant beschouwd worden, wat de resultaten robuuster maakt. Al met al is dit misschien een van de meest genoemde dingen om naar te kijken, en kan een goede plek om te beginnen zijn.

Irrelevante informatie weggooien

Soms raken LLMs in de war als je ze irrelevante informatie als context meegeeft, dan gaan ze dat gebruiken om de vraag te beantwoorden terwijl als je ze apart vraagt of het relevant is voor de vraag ze zullen zeggen dat die informatie dat niet is. Daarom is het geen heel slecht idee om ze dat apart te vragen, voor je de LLM aan de hand van de context de vraag laat beantwoorden kan je het eerst vragen wat van de gegeven context relevant is. Vervolgens kan je het vragen op basis van de relevante context, ipv de hele context, de vraag te beantwoorden. Dit wordt system 2 attention genoemd, en [hier](https://jrodthoughts.medium.com/inside-system-2-attention-meta-ai-new-method-to-improve-reasoning-in-llms-4424751a6be1) is daar meer over te lezen.

Opgave 2: Wat als je dingen wilt weten die niet in een paar stukjes tekst zijn

**Probleemstelling**

Zelfs als we altijd de meest relevante stukjes tekst vinden, zijn er vragen die niet met 2, 3 of 4 stukjes tekst die passen bij de vraag beantwoord kunnen worden. Op gegeven moment had ik een aantal gespreksverslagen (groter dan aantal chunks dat retrieved wordt) ingeladen en vroeg ik wie er in totaal in die gesprekken verhoord waren, dit is simpelweg een onmogelijke vraag voor onze architectuur omdat het antwoord daarop verdeeld staat in al die verslagen en niet in een paar relevante stukjes tekst te vinden is.

Voor de overheid is dit zeer relevant, zeker als we het willen gebruiken voor bijvoorbeeld wetgeving. In wetgeving staat de relevante tekst vaak niet in een alinea maar wordt voortdurend doorverwezen naar andere wetsartikelen waar verdere invulling wordt gegeven, om dus iets zinvols te kunnen zeggen moet die volledige keten gevolgd worden. Dit vereist meer dan één zoekopdracht.

**Opdracht**

Om het palet aan vragen dat door RAG beantwoord kan worden te vergroten zul je echt toevoegingen op de architectuur moeten maken. Dat maakt deze opdracht uitdagend, er zijn een aantal wat grotere ideeën over hoe dit zou kunnen waarbij het mij logisch lijkt er een te kiezen en te kijken of het je lukt daarmee vragen te beantwoorden die daarvoor onmogelijk waren voor de architectuur.

**Technieken**

Knowledge graphs

In een knowledge-graph probeer je de relatie tussen verschillende concepten (aan de hand van je documenten) in kaart te brengen, in het voorbeeld van de verhoren zou je bijvoorbeeld in kaart kunnen brengen wie er allemaal door wie verhoord wordt of bij wetgeving naar welke wetsartikelen er in andere wetsartikelen verwezen wordt. Vervolgens kan de informatie in de knowledge-graph (samen met de similarity search) gebruikt worden om vragen te beantwoorden. In dit [artikel](https://medium.aiplanet.com/implement-rag-with-knowledge-graph-and-llama-index-6a3370e93cdd) wordt dit uitgebreider beschreven. Bovendien kan je je LLM gebruiken om op basis van je documenten automatisch een dergelijke graaf te construeren, zoals [hier](https://medium.com/@milena.trajanoska/automated-knowledge-graph-construction-using-chatgpt-ba959050405a) beschreven.

Self-RAG

Het basisidee van Self-RAG is dat er niet meteen tekst bij de vraag gezocht wordt maar je eerst het model een antwoord laat generen en laat reflecteren op welke informatie nog mist voor een volledig antwoord, vervolgens gaat het voor die ontbrekende informatie een similarity search uitvoeren, en bepalen welke context die het terug krijgt wel en niet relevant is. Op basis hiervan genereert het model een nieuw antwoord en reflecteert opnieuw of er nog informatie ontbreekt om dit weer op te zoeken totdat alle relevante informatie gevonden is. Dit is heel nuttig voor bijvoorbeeld de wetgeving casus. Je kan je voorstellen dat je vraagt hoeveel belasting je in een specifieke situatie moet betalen hij in de eerste iteratie de situatie opzoekt om te zien dat je een bepaald type belasting moet betalen en de hoeveelheid in een andere wet gespecificeerd is om in de tweede iteratie die andere wet op te zoeken etc.

Dit goed vormgeven zou waarschijnlijk niet super eenvoudig zijn, maar er is [veel](https://arxiv.org/abs/2310.11511) [over](https://levelup.gitconnected.com/upgrade-your-retrieval-augmented-generation-with-self-rag-bb30b2a0ffa4) [geschreven](https://towardsdatascience.com/how-self-rag-could-revolutionize-industrial-llms-b33d9f810264) en [open source code](https://github.com/AkariAsai/self-rag) voor beschikbaar.

Overige technieken

**Probleemstelling**

Er zijn nog veel meer technieken waarmee RAG verbeterd kan worden, tot nu toe hebben we vooral gefocust op het ‘retrieval’ gedeelte. Daar is wellicht het meeste verbetering mogelijk, maar ook in andere delen van het proces zijn er verbeteringen te maken.

**Opdracht**

Dit is meer een lijstje van technieken die je op zou kunnen pakken om RAG te verbeteren, geen samenhangende opdracht met een specifiek doel.

**Technieken**

Query transformation

Wanneer we een RAG architectuur in productie zouden willen brengen moeten we er rekening mee houden dat de gebruiker geen perfecte en uitvoerige prompts gaat schrijven waar alle juiste trefwoorden in staan en de best geformuleerde instructies. Daarom kan het helpen stappen toe te voegen om het voor de gebruiker makkelijker maakt het systeem te gebruiken en voor het systeem om de gebruiker te begrijpen. [Query](https://docs.llamaindex.ai/en/latest/examples/query_transformations/query_transform_cookbook.html#) [transformations](https://towardsdatascience.com/a-guide-on-12-tuning-strategies-for-production-ready-rag-applications-7ca646833439#a5e2) is in feite een paraplu term voor manieren om de vraag van de gebruiker automatisch aan te passen zodat de RAG structuur die je gemaakt hebt beter werkt.

Een voorbeeld van query transformation die we al toepassen is het gebruik maken van een systeem prompt, een vaste prompt die je toevoegt bij ieder verzoek waarin bijvoorbeeld staat dat het antwoord gebaseerd moet zijn op de context en in het Nederlands moet. Deze ‘systeem prompt’ kan ook (met trial-and-error) nog verbeterd worden.

Bronvermelding verbeteren

Een van de grote krachten van RAG is dat het antwoord controleerbaar is, echter deze kracht heeft het meest effect als mensen ook daadwerkelijk de antwoorden controleren. In praktijk zal niet iedereen dit doen, zeker als het veel en onoverzichtelijk werk is. Laatst stonden we met onze applicatie op de winterfair, waar men vragen aan het systeem kon stellen waarbij niet alleen antwoorden maar ook bronnen gegeven werden, in praktijk was niemand geïnteresseerd in die bronnen. Dit kan beter door bijvoorbeeld de LLM zelf de stukken tekst te laten quoten waar die zich op baseert of de chunks overzichtelijker te maken en relevante zinnen zwartgedrukt, kan ook goede toevoeging zijn op andere technieken.

Embeddings of rerankers fine-tunen

Net zoals dat je een taalmodel kan fine-tunen op eigen data kan je ook embeddings of rerankers [fine-tunen](https://towardsdatascience.com/a-guide-on-12-tuning-strategies-for-production-ready-rag-applications-7ca646833439#341d) om beter te werken voor jouw use case (RAG) en soort documenten. Als ik de [resultaten](https://medium.com/better-programming/fine-tuning-your-embedding-model-to-maximize-relevance-retrieval-in-rag-pipeline-2ea3fa231149) zie lijkt mij dit niet het eerste om je tijd in te steken als je voornaamste doel is om snel veel verbetering te krijgen, maar het heeft zeker toegevoegde waarde dus als je ervaring met fine-tunen zoekt is dit een optie.

Chunking strategie verbeteren (semantisch chunken)

Je chunking strategie is bij RAG ook zeer relevant, wij hebben ge-experimenteert met verschillende lengtes en overlap van de stukken tekst waarin je documenten opgeknipt worden. Nu pakt het systeem zinnen samen tot het ongeveer op het juiste aantal tokens heeft om een chunk te maken, waarbij het rekening houdt met de opmaak van een document. Dus een chunk kan beter tussen alinea´s stoppen dan halverwege een alinea.

Dit kan echter verbeterd worden door puur naar opmaak en lengte te kijken maar ook naar betekenis. In feite laat je een LLM kijken welke tekst kwa betekenis bij elkaar hoort en dat worden je chunks. Dit heet [semantic chunking](https://medium.com/programmers-journey/three-open-source-rag-tools-you-need-to-know-about-331c3f28ab22) en [NEUM AI](https://github.com/NeumTry/NeumAI) is een open source tool om dit te doen.

**Conclusie**

Dit is geen definitie lijst aan verbeteringen die mogelijk zijn, er zullen in de komende maanden vast ook veel nieuwe ideeën bedacht worden dus voel je ook vrij zelf met iets anders aan de slag te gaan. Hopelijk biedt deze lijst echter wat inspiratie en zit er iets tussen waarmee je aan de slag kan. Overigens hoef je niet alles van 0 zelf te maken, als je ergens wat code ziet die je over kan nemen en het werkend krijgt voor onze applicatie (of iig met een lokaal gedownloade open source LLM), is dat precies even goed. Hier wat repo’s waar mogelijk relevante code in staat:

* [Weavite](http://weaviate/Verba:%20Retrieval%20Augmented%20Generation%20(RAG)%20chatbot%20powered%20by%20Weaviate%20(github.com))
* [Azure search](http://azure-samples/azure-search-openai-demo:%20A%20sample%20app%20for%20the%20Retrieval-Augmented%20Generation%20pattern%20running%20in%20Azure,%20using%20Azure%20AI%20Search%20for%20retrieval%20and%20Azure%20OpenAI%20large%20language%20models%20to%20power%20ChatGPT-style%20and%20Q&A%20experiences.%20(github.com))
* [Semantic Kernel](http://microsoft/semantic-kernel:%20Integrate%20cutting-edge%20LLM%20technology%20quickly%20and%20easily%20into%20your%20apps%20(github.com))

1. <http://boston.lti.cs.cmu.edu/luyug/HyDE/HyDE.pdf> [↑](#footnote-ref-0)
2. Hier twee voorbeelden van deze strategie toegepast: [Auto Merging Retriever - LlamaIndex 🦙 0.9.22](https://docs.llamaindex.ai/en/latest/examples/retrievers/auto_merging_retriever.html#), [Parent Document Retriever | 🦜️🔗 Langchain](https://python.langchain.com/docs/modules/data_connection/retrievers/parent_document_retriever), zie ook 2.4 en 2.4.2 hiervan: [Advanced RAG Techniques: an Illustrated Overview | by IVAN ILIN | Dec, 2023 | Towards AI](https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6) [↑](#footnote-ref-1)