# RAG en implicaties (Learning Lion)

*Een explainer van de technologie (RAG) die gebruikt wordt voor de use-case Kamervragen van JenV*

Om verwarring in terminologie te voorkomen, volgt hier een korte samenvatting van hoe men RAG en zijn werking kan conceptualiseren.

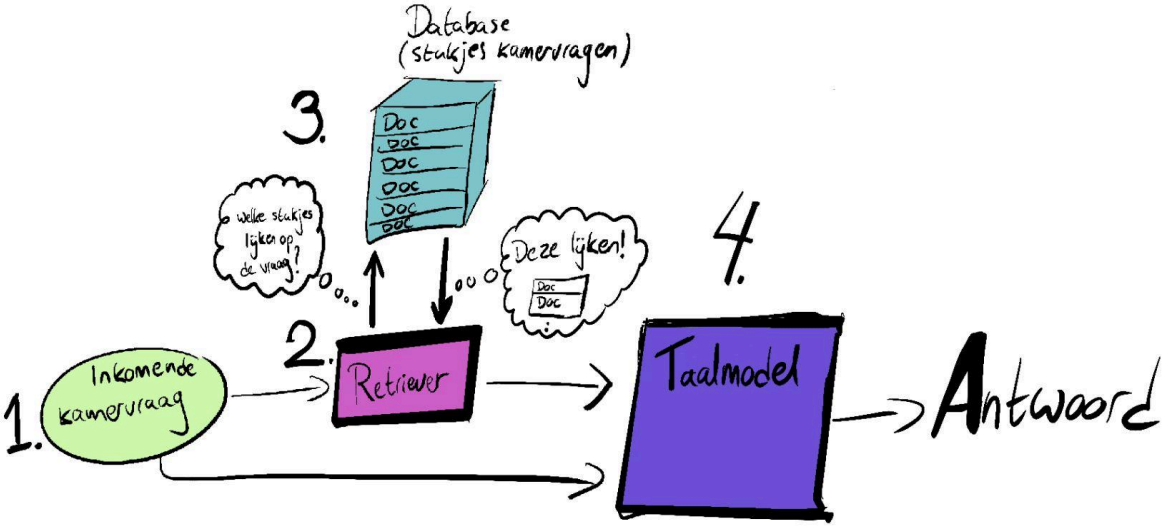
**Disclaimer:** De concepten uitgelegd in onderstaand stuk zijn abstracties van de exacte werking, waardoor nuances kunnen ontbreken. Toch dient de volgende explainer hopelijk als vertrekpunt voor de vormgeving van de databehoefte en de vormgeving van de use-case Kamervragen door JenV. Let op! De data waarover hier gesproken wordt, wordt niet gebruikt om de technologie mee te trainen. Hier wordt later dieper op ingegaan.

## Overview: Wat is RAG?

RAG staat voor **Retrieval Augmented Generation**. Er zijn twee kerntaken die een RAG-systeem (pipeline) dient uit te voeren:

1) **Retrieval**: Het ophalen van relevante documenten uit een database die zich verhouden tot een vraag (of ander soort tekstuele input).
2) **Generation**: Op basis van die relevante documenten een antwoord op een vraag (of andersoortige tekstuele input) vorm te geven.

Visueel gezien ziet een basis RAG-pipeline er als volgt uit:



1. De vraag die het systeem binnenkomt. In de huidige vorm is dat een subvraag uit een document ‘Kamervragen met beantwoording’, welke is vertaald naar numerieke waarden.
2. **Retriever**: Hier wordt de inkomende vraag semantisch vergeleken met stukjes tekst uit de database.
3. Database: Een datastructuur waarin stukjes tekst uit eerdere kamervragen en beantwoording zijn opgeslagen. Dit vormt de knowledge-base van de RAG-pipeline.
4. **Taalmodel**: een Large Language Model (LLM) die tekst genereert op basis van de context (de zojuist gevonden relevante stukjes tekst uit de database) met de inkomende kamervraag. Deze krijgt instructies om zich te beperken tot de vraag en deze context.

## Een laag dieper

Onder de motorkap van de visualisatie zijn veel verschillende variabelen aanwezig die invloed kunnen hebben op het resultaat uit de RAG-pipeline (output). Een paar van deze variabelen die wij belangrijk achten voor de huidige use-case worden op het moment getest en zijn onderstaand simpel uiteengezet:

1. De inkomende vraag in kwestie: Wij voorzien dat dit een subvraag is uit een document van type ‘Kamervragen met beantwoording’ met eventueel toegevoegd de zgn. ‘Inleiding’. De inleiding bevat veel informatie over het doel van de vraag. Deze informatie ontbreekt vaak in de losse subvragen. Door de inleiding toe te voegen, vermoeden wij beter relevante documenten op te kunnen halen.
2. **Retriever**: Op semantisch niveau tekst vergelijken gaat middels het vergelijken van getallenreeksen. Er is dus een vertaalmachine nodig van tekst (input) naar getalreeksen. Zo’n vertaalmachine heet ook wel een embeddingsmodel of tokenizer (voor meer informatie, zie A). Deze getallen zijn als het ware coördinaten van betekenis van stukken tekst. Als het model goed werkt zullen er ook documenten opgehaald worden die semantisch dicht bij elkaar liggen en niet slechts documenten met precies dezelfde zoekterm. Denk aan bijvoorbeeld documenten met het woord “kat” en een document met het woord “korthaar”.
3. Database (of vector store): De tekst in de database is door dezelfde vertaalmachine (het embeddingsmodel) vertaald. Daardoor kunnen de documenten in de database worden vergeleken met de vraag. De tekst van de documenten kan gesplitst zijn in grotere of kleinere stukjes, afhankelijk van wat goed werkt voor de use case.
4. **Taalmodel**: een Large Language Model (LLM) die tekst genereert op basis van de context (de zojuist gevonden relevante stukjes tekst uit de database) met de inkomende kamervraag. Deze krijgt instructies om zich te beperken tot de vraag en deze context.

De modellen waarover wordt gesproken zijn voorgetrainde taalmodellen. Dit betekent dat deze vooralsnog **niet** worden *getraind* op de kamervragendata. De modellen worden puur ingezet als vertaalmachine (embedding model) of generator (LLM) op basis van hun eerder opgedane ‘kennis van taal’ met de kamervragendata als *context* of ‘kennis’.

# Nog een laag dieper

## A) Embeddingsmodel (of tokenizer)

Hoe vertaalt een machine tekst naar een format waar deze niet alleen de syntax van de tekst vastlegt, maar ook semantische aspecten meeneemt, zodat tekst flexibeler vergeleken kan worden?

**Achtergrond**
Het embedding model wordt getraind op grote hoeveelheden tekst.[1] Het model kan bijvoorbeeld voorspellingstaken uitvoeren om het te trainen. Denk aan het raden van woorden. Elke keer als het model dit fout heeft krijgt het een signaal terug dat de voorspelling net een beetje moet worden aangepast.

Stel je de volgende zinnen voor:

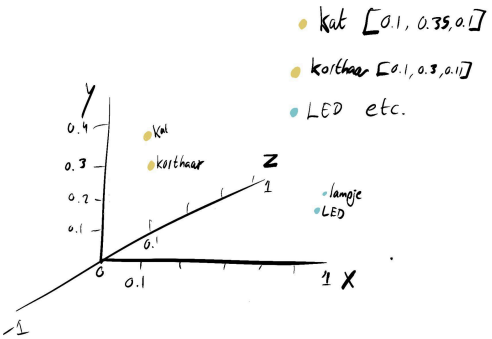
a) *'De europese korthaar is een gedomesticeerd huisdier'*
b) *'Ik heb een kat als huisdier'*

Als we nu 'korthaar' en 'kat' weghalen, zal het voor ons mensen intuïtief uit de context van de woorden af te leiden zijn dat 'kat' en 'korthaar' qua betekenis 'dicht bij elkaar liggen' en dat 'lampje' een minder voor de hand liggend antwoord zal zijn. Tijdens de training vragen we het model het missende woord te voorspellen. Door feedback wordt het model steeds beter in deze woordgroepjes vorm te geven.

**Intuïtie**
Stel je een virtuele ruimte voor in 3D. In alle drie de dimensies bevinden zich groepjes woorden die semantisch gezien dicht bij elkaar liggen, bijvoorbeeld een groepje met 'lampje' en 'LED' en eentje met 'kat' en 'korthaar'. Die woorden kunnen coördinaten meekrijgen om aan te geven waar in die ruimte zij zich bevinden (namelijk langs een x,y,z-as). Het embeddingsmodel geeft deze coördinaten aan de woorden.

Als men dit model dan gebruikt om te zoeken naar woorden die iets te maken hebben met 'korthaar', is het mogelijk dat er ook documenten naar voren komen met 'kat' en 'hond'. Deze woorden liggen namelijk dicht bij elkaar in de virtuele ruimte. Het model hoeft alleen maar naar de dichtstbijzijnde getalreeksen te zoeken.

In de praktijk kan er op verschillende tekstgroottes (zin, woord, woorddeel, letter) numerieke coördinaten worden vormgegeven. Dit hangt af van de achtergrond van het model. Het is uiteraard niet precies zo simpel als beschreven. Daarnaast geldt bij dit soort modellen typisch niet een 3D




[1] Wij maken gebruik in onze RAG-pipeline van een bestaand embeddingsmodel. Wij trainen dus zelf het embeddingsmodel niet.

representatie, maar een representatie van wel tientallen of honderdtallen dimensies. Dit is echter moeilijk voor te stellen.[2]

## B) Taalmodel

Als de stukjes tekst opgehaald zijn uit de database door de retriever, mag het taalmodel aan de slag. Deze is ook voorgetraind op grote hoeveelheden tekst.[3] In het geval van RAG, vaak op vragen en antwoorden, zodat het model goed is in het voorspellen van taal.

Het taalmodel in onze RAG-pipeline zal tekst genereren op basis van de context (de stukjes tekst die van belang zijn voor het beantwoorden van de vraag) en de vraag.

Als een taalmodel niet getraind is op kamervragen, zit de specifieke kennis van de kamervragen en hun inhoud typisch niet ingebakken in de meeste taalmodellen op detailniveau. Zelfs als een taalmodel wel kamervragen als onderdeel van de trainingsdata ‘gezien’ heeft, is de kans op hallucineren aannemelijk.

Sommige taalmodellen kunnen goed instructies opvolgen. Wanneer ze specifiek de taak krijgen om materiaal te genereren op basis van een vraag en expliciete tekst uit een eigen database, is (intuïtief) de kans op hallucinaties kleiner en de kans op feitelijk juiste informatie in een ander jasje (bijvoorbeeld uit meerdere bronnen samengevat) groter.

[2] Deze intuïtie is onder andere geïnspireerd op Deciphering the Dimensions of Embeddings: A Journey into Semantic Spaces | by Anand | Medium.
[3] Ook hier gebruiken we dus een reeds bestaand model en gebruiken we de aangeleverde data niet om te trainen.

# Overview: Data scope

Door het logische gebrek aan affiniteit met RAG aan de kant van JenV ontstond er verwarring over welke data nu voor wat wordt gebruikt. Nu de basis van RAG is verduidelijkt is het belangrijk voor JenV om de scope van de data te bepalen.

Belangrijke aandachtspunten:

- De database: Wanneer de retriever een speld in een hooiberg mag zoeken, is het makkelijker om een speld te vinden in een kleinere hooiberg dan in een grote. Wanneer de database van eerder gestelde kamervragen groot is, zal de ruis voor de retriever toenemen en de relevantie van de opgehaalde tekststukjes in de database afnemen. Wel is het belangrijk dat de hooiberg groot genoeg is zodat de externe validiteit van de RAG-pipeline wordt gewaarborgd.

Tip: Baseer de database van de vragen op de zoekwijdte die het meest wordt gebruikt door domeinexperts.

**Vraag:** Hoe groot wordt de brondata waarop de database wordt gebaseerd? (bijv. een half jaar aan kamervragen met beantwoordingen)

- De Inkomende kamervraag: De dataset die werd voorgesteld om steeds als een inkomende kamervraag te fungeren bestaat uit 50 eerder gestelde sub-vragen. Omdat bij nieuwe kamervragen vaak een exact correct antwoord ontbreekt, is het mogelijk interessant voor de externe validiteit van de test om deze subvragen anders te formuleren.

Tip: als we besluiten met subvragen te werken kan de inleiding van de set vragen van cruciaal belang worden voor een correct antwoord. Deze wordt dan mogelijk gevraagd om ook op te geven in de uiteindelijke tool.

**Vraag:** hoe worden vragen bewerkt om de externe validiteit van de tool te testen?