**Academiejaar**

2023 - 2024

Large Language Models

LLM Chatbots voor Domein Specifieke Vragen

Vincent Verbergt

Bachelorproef tot het behalen van een diploma Bachelor of Science in de

**industriële wetenschappen: elektronica-ICT**

Promotoren

Prof. dr. Peter Hellinckx

Inhoudstabel

[1.1 Van Sequenties Tot Chatbots 3](#_Toc165377714)

[1.2 Domein Specifieke Chatbots 4](#_Toc165377715)

[1.3 Trainen Van Chatbots 5](#_Toc165377716)

[1.4 Modellen Vergelijken 5](#_Toc165377717)

[2.1 Benchmarks selecteren 6](#_Toc165377718)

[2.2 Model Analyse 6](#_Toc165377719)

[2.2.1 Open-source 6](#_Toc165377720)

[2.2.2 Closed-source 9](#_Toc165377721)

[4.1 Finetuning 10](#_Toc165377722)

[4.2 RAG 10](#_Toc165377723)

# Abstract

We onderzoeken de mogelijkheden van Large Language Models om een domein specifieke kennisbron te onderzoeken en behulpzaam vragen te beantwoorden. De methodes die we onderzoeken zijn finetuning en Retrieval Augment Generation (RAG). De prestaties van de systemen vergelijken we aan de hand van een benchmark en een concreet examen. [TODO aanvullen met resultaten en conclusie]

# Inleiding

Met de publicatie van OpenAI’s ChatGPT (GPT3.5) in het najaar van 2022 is er een explosie binnen de AI wereld. Meer specifiek het Natural Language Processing (NLP) veld en lijkt dit momenteel de beste koers richting algemene artificiële intelligentie. De modellen zijn getraind op grote informatiebronnen van het internet en houden een solide basiskennis van alle domeinen. Dit maakt hen *Jacks of all trades but master of none*.  
Hebben de LLM’s het in zich om toch expertise te vergaren in een specifiek veld?

Generatieve AI komt ook vaker en vaker voor in de support systemen van AI. Een perfect voorbeeld hiervan is Kate AI van KBC. Een ander veld waar dit ook voorkomt is in documentatie van code. [Serverpod’s](https://docs.serverpod.dev/) Documentatie heeft een ingebouwde assistent die perfect de staat van het project kent. Het is technologie Kapa.ai die Retrieval Augmented Generation (RAG) gebruikt.

Retrieval Augmented Generation is een oplossing op het probleem van snel een generatieve AI kennis op te doen. In deze methode neem je de vraag van de gebruiker en vergelijk je dit met de volledige kennisbron. Alle relevante passages voeg je toe aan de finale prompt van de assistent die deze dan mooi samenvat.  
(<https://arxiv.org/abs/2005.11401>)

RAG is zeker niet de enige manier om kennis te vergaren. Als we de dominante GPT-4 een vraag stellen is het duidelijk dat deze een brede basiskennis heeft. Deze kennis werd opgedaan tijdens de trainingsfase van het model. Deze trainingsfase vergt gigantisch veel data en riskeert de kans dat het geen taalbegrip verkrijgt door een te kleine hoeveelheid data. Wat er wel gedaan wordt, is een voorgetraind model finetunen. Dit wil zeggen een kleine hoeveelheid training op het model doen met de specifieke kennis.

Beide methodes hebben hun voor en nadelen. De verwachte resultaten zullen zijn dat een gefinetuned model kleiner is en daardoor een snellere inferentie tijd heeft. RAG vereist een grote attentiespanne / context-window. Bij deze methode moeten ook de voltallige kennis verwerkt worden door het model, dit is tijd dat het model geen antwoord aan het genereren is.

Om de resultaten effectief te kunnen gaan vergelijken moeten we een representatieve benchmark opstellen. Deze benchmark moet niet enkel kennis van het systeem testen maar ook de vaardigheden als assistent. Dit houdt in dat het onzekerheid kan uitdrukken en vragen niet beantwoorden die niet in het domein liggen. Ook willen we zien of de systemen net 1 stap verder kunnen denken en hun deductie vaardigheid testen.

# Modelkeuze

De proef heeft 3 doelen:

1. Kan een Chatbot model expertise vergaren in een specifiek domein
2. Welke methodes werken en welke is beter
3. Zijn er bepaalde architecturen die beter presteren in een bepaalde methode.

Om hieraan te beginnen moeten we voor de finetuning en RAG methode modellen kiezen. Om een beeld te krijgen op het huidige landschap kunnen we een data-analyse op benchmark leaderboards uitvoeren. Voordat we deze analyse kunnen uitvoeren moeten we weten welke eigenschappen we graag als fundament hebben en welke benchmarks hierop testen.

## Benchmarks selecteren

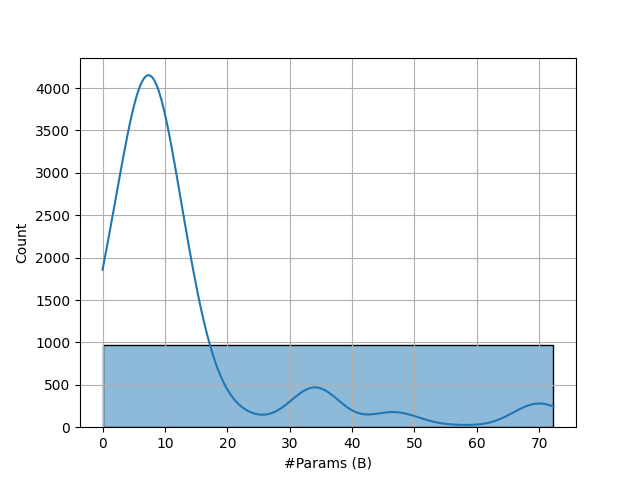
Als basis van onze modellen, ongeacht de methode waar ze in toegepast worden, zoeken we vooral comprehensie van de taal. Veelgebruikte benchmarks die deze vaardigheden testen zijn de HellaSwag en Winogrande benchmark. Deze testen zijn zeer makkelijk voor mensen, maar gigantisch complex voor taalmodellen.

## Model Analyse

Er wordt een onderscheid gemaakt tussen open-source en closed-source. Open-source modellen kun je altijd finetunen aangezien de volledige structuur gekend is. Closed-source modellen laten dit niet altijd toe. Ook kun je ze niet zelf hosten en moet je altijd via hun API’s werken tegen een bepaalde kost. Dit is een redelijke consequentie aangezien de hoge hardware kosten die je moet maken om deze modellen te laten werken.

### Open-source

Voor open-source benchmarks is het [HuggingFace leaderboard](https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard) de beste informatiebron. Een eerste analyse die gemaakt wordt is zien hoe de modelgroottes verdeeld worden. In de frequentieplot van Figuur 1 zien we 3 categorieën verschijnen. De lijnen dat ze verdeelt liggen op 25 miljard en 58 miljard parameters.



Figuur 1 Grafiek modelgrootte frequentie.

Per categorie rangschikken de modellen zich op de gemiddelde score tussen de HellaSwag en Winogrande benchmark. Dit doen we omdat beide scores voor ons dezelfde relevantie hebben en een hoge score in de ene test correleert vaak met een hoge score in de andere. Deze resultaten zijn te vinden in onderstaande tabellen.

|  |  |  |  |
| --- | --- | --- | --- |
| Architectuur | **HellaSwag (%)** | **Winogrande (%)** | **Model** |
| MixtralForCausalLM | 89.02 | 89.27 | zhengr/MixTAO-7Bx2-MoE-Instruct-v6.0 |
| MixtralForCausalLM | 89.72 | 87.85 | rizla/rizla-17 |
| MixtralForCausalLM | 89.3 | 88.24 | yunconglong/Truthful\_DPO\_TomGrc\_FusionNet\_7Bx2\_MoE\_13B |

Tabel 1 Top 3 scores modellen [0, 25[ miljard parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| Architectuur | **HellaSwag (%)** | **Winogrande (%)** | **Model** |
| CohereForCausalLM | 87.96 | 83.82 | CohereForAI/c4ai-command-r-plus |
| MixtralForCausalLM | 87.73 | 82.56 | Swisslex/Mixtral-8x7b-DPO-v0.2 |
| MixtralForCausalLM | 87.13 | 82.95 | Steelskull/Lumosia-MoE-4x10.7 |

Tabel 2 Top 3 scores modellen [25, 58[ miljard parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| Architectuur | **HellaSwag (%)** | **Winogrande (%)** | **Model** |
| LlamaForCausalLM | 89.77 | 87.53 | MTSAIR/MultiVerse\_70B |
| LlamaForCausalLM | 88.6 | 85.4 | Undi95/Miqu-70B-Alpaca-DPO |
| LlamaForCausalLM | 88.61 | 85.32 | 152334H/miqu-1-70b-sf |

Tabel 3 Top 3 scores modellen 58+ miljard parameters.

|  |  |  |
| --- | --- | --- |
| Architectuur | **Gemiddelde (%)** | **#Params (Miljard)** |
| MixtralForCausalLM | 87.20 | 24.15 |
| CohereForCausalLM | 86.98 | 45.015 |
| MistralForCausalLM | 85.46 | 7.24 |

Tabel 4 Top 3 scores per architectuur.

De resultaten laten zien dat Mixtral de winnaar is voor de kleine modellen ( < 58 miljard parameters). We zien dat Llama de winnaar is bij de grotere open-source modellen. Ook valt op dat de verschillen categorieën de prestaties dicht bij elkaar liggen.

Aangezien finetuning zeer GPU intensief kiezen we best zo’n klein mogelijke modellen. Tabel 4 laat zien dat Mixtral verre uit de beste is. De verschillende modellen zijn te vinden op [HuggingFace ’s MistralAI pagina](https://huggingface.co/mistralai). Wat we hier zien is dat de grootte van het Mistral model aanzienlijk kleiner is en weinig in prestatie verliest. Dit model is ideaal om de finetuning methode op uit te testen.

### Closed-source

In Tabel 5 staan dezelfde gegevens maar dan voor de closed-source modellen. Een opmerking dat gemaakt moet worden is dat deze modellen niet dezelfde globale beschikbaarheid hebben. Hierdoor zijn bijvoorbeeld Google ’s Gemini modellen niet beschikbaar in Europa.

|  |  |  |
| --- | --- | --- |
| Model | **HellaSwag (%)** | **WinoGrande (%)** |
| GPT-4 | 95.3 | 87.5 |
| GPT-3.5 | 85.5 | 81.6 |
| Claude-3 Opus | 95.4 | 88.5 |
| Claude-3 Sonnet | 89.0 | 75.1 |
| Claude-3 Haiku | 85.9 | 74.2 |
| Gemini 1.0 Ultra | 87.8 | / |
| Gemini 1.5 Pro | 92.5 | / |
| Gemini 1.0 Pro | 84.5 | / |

Tabel 5 Scores closed-source modellen ([source](https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf)).

Het beste model is Claude-3 Opus, een multimodaal model van Anthropic. Ook zien we dat GPT-4 van OpenAI heel hoog scoort. Gemini’s prestaties lopen helaas achter op de state-of-the-art. De grootste drijfveer van de verhoogde prestaties zijn de groottes van de modellen. Hoewel de exacte specificaties niet bekend zijn zitten deze modellen ruim in de biljoen parameters. Dit gaat helaas gepaard met verhoogde inferentie kosten.

<https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf>

# Systeem evaluatie

Om de verschillende implementaties met elkaar te kunnen vergelijken moet er een benchmark zijn. De beste manier om veel resultaten te halen uit Natural Language Processing systemen is met behulp van een meerkeuze test. Deze heeft het voordeel dat je de antwoorden snel kunt evalueren maar laat weinig ruimte voor genuanceerde antwoorden.

<https://arxiv.org/abs/2308.09975>

<https://openreview.net/forum?id=_VjQlMeSB_J>

Uit onderzoek blijkt dat wanneer een meerkeuze test wordt gebruikt, er betere antwoorden gegenereerd worden door een uitleg van de keuze te vragen. Dit is de Chain-of-Thought (CoT). Dit kan ervoor zorgen dat de modellen beginnen te redeneren en deductie gaan vertonen.

Een keuze dat gemaakt moet worden wanneer we een NLP gaan testen is of er voorbeeldvragen meegegeven worden. De term hiervoor is N-shotting, waar N de hoeveelheid voorbeeldvragen is. Typische waardes voor N zijn 0, 5, 10, 20. Voor bepaalde waardes gaan de testscores omhoog maar dit is afhankelijk van de test en het model. Aangezien de toepassingsomgeving van de systemen een zero-shot is gaan we de testen ook zo uitvoeren.

De vragen waaruit de zero-shot CoT meerkeuze benchmark bestaat, moeten representatief zijn aan wat we willen testen. De benchmark moet kennis, redenering, hallucinatieresistentie en domeingerichtheid testen. Daarom bestaat de benchmark uit volgende categorieën vragen:

* Letterlijk:
  + Deze vragen kunnen beantwoord worden met letterlijke passages uit de kennisbron.
  + 13 vragen.
* Deductie:
  + Deze vragen vereisen een extra denkstap. Hiermee testen we redeneringsvermogen.
  + 3 vragen.
* Fout:
  + Deze vragen bevatten enkel foute antwoorden. Hiermee kunnen we hallucinaties opvangen.
  + 4 vragen.
* Buiten Domein:
  + Deze vragen vallen buiten het domein en testen domeingerichtheid.
  + 3 vragen.

Elk systeem testen we 5 keer omdat er inherente variantie zit op de testscores. De systemen zijn namelijk niet determinant en vertonen willekeurig verdrag. Om dit in kaart te brengen en een meer genuanceerd beeld van de systemen te verkrijgen gaan we ook open vragen geven die manueel verbeterd worden.

# Implementatie

## Finetuning

Finetunen en trainen van LLM’s is in het algemeen een zeer hardware intensieve taak. De voornaamste bottleneck is de geheugenvereiste van deze modellen. Nemen we bijvoorbeeld het Mistral 7 miljard F16 model dan is de grootte van het model het volgende:

Bij het trainen van een Transformer model moet je effectief de grootte van het model \* 4 doen. Dit resulteert in een geheugen vereisten 56 GB. Om dit te kunnen halen moeten we gaan trainen met Cloud resources. Hiervoor werd [AutoTrain](https://huggingface.co/autotrain) gebruikt. Dit voorzag ons van een ruimte waar we modellen konden finetunen aan de hand van datasets.

Nadat het model gefinetuned, heb ik het [gedownload](https://huggingface.co/vincentverbergt/Mistral7B-DIP) en via de [transformer library](https://huggingface.co/docs/hub/transformers) in python code ingeladen. Om de inferentietijd te verhogen heb ik het model gekwantiseerd naar 1/4de van de originele grootte. Hierna heb ik een pipeline gecreëerd om tekst te sturen naar het model. Om te zorgen dat het model de testvereisten kenden heb ik nog enkele prompts toegevoegd die het domein aangeven en de meerkeuze test formaat uitlegt.

## RAG

# Resultaat

# Conclusies



<Quote> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris efficitur diam sit amet viverra lacinia. In porttitor diam ex.

|  |  |  |  |
| --- | --- | --- | --- |
| Tabelkop | Tabelkop | Tabelkop | Tabelkop |
| Tabelinhoud | Tabelinhoud | Tabelinhoud | Tabelinhoud |
| Tabelinhoud | Tabelinhoud | Tabelinhoud | Tabelinhoud |

<Korte tekst>Sed quis ante ut erat placerat condimentum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae

## Van Sequenties Tot Chatbots

De origine van de moderne chatbots is te vinden in het sequentie naar sequentie (seq2seq) probleem. Hierin proberen we een reeks van variabele lengte invoer te transformeren in een variabele lengte uitvoer. Een goed voorbeeld hiervan is zijn vertalingen van teksten maar ook kan dit een proteïneketen omzetten naar een 3D-structuur zijn. Dit lijkt toch nog niet helemaal op de vorm van een chatbot, maar hieronder illustreer ik hoe dezen met elkaar verbonden zijn.

Een eerste aanpak voor het seq2seq probleem is het gebruiken van Recurente Neurale Netwerken (RNN). Deze lijken op feed-forward netwerken maar de RNN’s hebben een extra invoer, namelijk de uitvoer van de iteratie. Hiermee kunnen we meerdere invoeren reduceren naar een vast aantal getallen (context-vector), dit is de coderingsfase. De decodering lijkt op de codering, maar nu is de initiële staat van het systeem de context-vector. We geven dan een speciale input aan het systeem, bijvoorbeeld het EOS token, en laten dit 1 uitvoer genereren. In de volgende iteratie wordt deze uitvoer dan de invoer en genereren we een extra uitvoer. Dit blijven we herhalen tot een stopconditie bereikt wordt.

Maar hoe kan een systeem dat met getallen werkt taal en andere zaken gebruiken? Hiervoor moeten we onze invoer eerst door een vector embedding systeem laten processen. Belangrijk aan dit systeem is dat het al bepaalde eigenschappen van de invoer kan opvangen en weergeven. In woord-embedding betekent dit concreet dat woorden met gelijkende betekenissen ook gelijkende waardes krijgen. Ook een eigenschap van taal is dat eenzelfde woord meerdere betekenissen kan hebben naargelang de context. Voor taal betekent dit dat we verschillende waardes genereren per woord om dit op te vangen.

Nu blijkt dat de meest eenvoudige vorm van dit systeem niet goed werkt voor taalprocessen. Er zijn hier 2 redenen voor. De RNN’s verliezen de invloed van invoer vrij snel en vertalingen gebeuren niet woord per woord. De oplossingen hiervoor zijn Long-Term-Short-Term Memory (LTSM) en attentie.

De reden waarom RNN’s de invloed van initiële invoer snel verliezen is omdat de output / interne-staat in elke iteratie vermenigvuldigd wordt met een vaste waarde. Hierdoor ontstaat een exponentieel verband waardoor in lange teksten de eerste woorden helemaal vergeten worden. Dit is belangrijk voor zinnen zoals “Geen katten eten”. LTSM’s lossen dit probleem door een nieuwe interne staat toe te voegen die additief is. Hierdoor verliezen we geen gradiënten op lange termijn.

Bekijken we de zin: “Ik hou van jou!”, en diens Franse vertaling “Je t’aime” dan zien we dat het lijdend voorwerp van plaats verandert. Men merkt dat zonder attentie-mechanismes, de context-vector niet genoeg informatie bevat om deze omwisseling te doen. Het doel van een attentie-mechanisme is om aan elke decoder iteratie, de invoer dat belangrijk is mee te geven. Zo zou de attentie van de 1ste iteratie op het woord “Ik” staan en in de 2de iteratie niet op het woord “hou” maar op “ jou”.

In de paper “Attention Is All You Need” ontdekte men dat een goed attentie mechanisme zodanig krachtig is dat we de RRN / LTSM modules niet meer nodig hebben. Daarbovenop gaan we in de coderingsfase van een sequentieel proces naar een parallel proces. Hierdoor kunnen modellen veel sneller werken maar waarschijnlijk nog belangrijker, ze kunnen veel sneller trainen. Door deze versnelde trainingstijd kunnen de modellen veel grotere datasets verwerken. Deze architectuur heet de Transformer architectuur en was een gigantische doorbraak voor het domein van Large Language Models (LLM).

Wat is nu de invoersequentie voor Chatbots? De uitvoer moet een transformatie zijn van invoer en net dit seq2seq idee botst met de werking van Chatbots, generatie van tekst. De coderingsfase verwerkt invoer en de decoderingsfase creëert uitvoer. Stel we nemen gebruiken enkel de decoderingsfase en we geven als eerste invoer onze vraag/prompt. Dan kunnen we het model uitvoer laten blijven genereren tot die een stop conditie bereikt. Dit heet de decoder-only transformator architectuur en vormt de basis van de meeste chatbots.

## Domein Specifieke Chatbots

Om een domein specifieke chatbot te maken zijn er veel technieken. Concreet gaan we een systeem maken dat vragen kan beantwoorden vanuit een kennisbron. Deze bot zou de vraag correct moeten kunnen interpreteren en in natuurlijke taal antwoorden. Ook hopen we dat het systeem bepaalde redeneringen/deducties kan maken. Belangrijk hierbij is, is een deductie correct is en het model ook laat weten dat het een afleiding is van kennis.

In het algemeen zijn er 3 methodes om dit gedrag te bereiken m.b.v. LLM’s:

1. **Ground-up:** We ontwerpen of kiezen een bestaande architectuur en trainen deze op een dataset.
2. **Finetunen:** We kiezen een bestaande, vooraf getrainde architectuur en trainen deze op een dataset.
3. **Retrieval Augmented Generation (RAG):** We maken gebruik van bestaande chatbots en voeden deze met de relevante kennis wanneer we de vraag stellen.

Bij de Ground-up aanpak gaan we een architectuur ontwerpen / kiezen en de parameters willekeurig initialiseren. Dit model willen we dan trainen op onze kennisbron en daarna finetunen om als chatbot / Q&A machine te werken. Deze methode zorgt ervoor dat de kennis het meest inherent is en heeft daardoor de grootste kans om correcte deducties te maken. Al is dit niet altijd waar omwille dat het model geen kennis gaat hebben in domeinen die het nodig heeft om deducties te maken. Ook moet de kennisbron zodanig groot zijn zodat het model een fundamenteel inzicht in taal comprehensie heeft. Deze aanpak vereist het meeste training en is ook het meest intensief. Als je elk van deze struikelblokken weet te overkomen dan kan dit misschien het beste resultaat opleveren.

Wanneer we finetunen kiezen we een (open-source) model dat al getraind is geweest. Dit betekent dat het model al een fundamenteel inzicht in de taal heeft en enkel nog maar kennis ontbreekt. Dit kunnen we oplossen door een training te starten op de kennisbron zonder de parameters te resetten. Als resultaat blijft er dan een chatbot over die over een inherente kennis van het domein beschikt op een veel kortere tijd. De struikelblokken hier zijn dat we enkel open-source modellen kunnen gebruiken. Ook beschikt het model waarschijnlijk over kennis uit andere domeinen, dit kan handig zijn voor betere deducties maar we wensen vaak dat het model enkel voor het domein gebruikt wordt en niet voor andere vragen. Een soort van filter moet dus geplaatst worden.

In de RAG methode gaan we de prompt ontleden en daarmee de stukken die met de vraag hebben uit onze kennisbron halen. Deze fragmenten geven we dan samen met de vraag aan de chatbot die de instructies heeft om met deze informatie een coherent antwoord te geven. Deze optie vereist geen intensieve training en is het meest bestendig tegen foute deducties. Dit komt omdat het alle informatie krijgt en dan zelf kan uitmaken of het genoeg kennis heeft om de vraag te beantwoorden. Ook kunnen we hierbij gebruik maken van zowel open- en close-source modellen. Dit is een voordeel omdat de laatsten huidig beter presteren op de standaard benchmarks.

## Trainen Van Chatbots

Hoe kun je de complexe architectuur van transformers nu trainen zodat het taal kan leren en kennis opneemt? Het proces werkt nagenoeg gelijk aan de klassieke neurale netwerken (NN) trainen. NN’s zijn functie-benaderaars, het doel is om a.d.h.v. concrete datapunten te leren wat er tussen deze punten ligt, generalisatie. De functies die we willen benaderen zijn vaak hoog-dimensionaal met meerder invoer en uitvoer.

De training van zo’n NN loopt als volgt: We geven het model een invoer van de dataset, we meten de uitvoer van het model en vergelijken deze met het verwachte resultaat, dit is de fout Daarna berekenen we de invloed van elke parameter van het model op de fout, dit is de gradiënt. De gradiënt vertelt ons hoe we elke waarde moeten aanpassen zodat de fout kleiner wordt. Dit proces herhalen is de training van een Neuraal Netwerk.

Een LLM trainen werkt hetzelfde maar de vraag is dan wat zijn onze verwachte waardes en wanneer kunnen we de fout evalueren. Specifiek voor de decoder-only architectuur kunnen we gebruik maken van de kennis dat het model probeert een ‘logisch’ woord probeert te geven aan de hand van wat er al gegeven is. Laten ons een voorbeeld doornemen:

We trainen een chatbot op de zin “De kat is harig”. We geven het woord “De” als input van het model en meten dat de output “boom” is. We kunnen hier ook de fout op meten door classificatie fout methodes. We berekenen gradiënten en houden deze voorlopig bij. Nu steken geven we het woord “kat” aan het model en het model leidt correct af uit “De kat” dat het volgende woord “is” moet zijn. We doen dit proces tot de volledige invoer verwerkt is en veranderen dan de parameters aan de hand van de berekende gradiënten.

Graag geef ik ook graag de schaal mee waarop we werken. Ook al is de exacte grootte van OpenAI’s GPT4 model niet geweten, wordt geschat in de grootorde van een biljoen parameters. Deze modellen worden in gespecialiseerde datacentra getraind over een periode van +- 90 dagen. Het werk wordt verdeeld over zo’n 8000 peperdure GPU’s. Zou je je eigen GPT4 model willen trainen op een consumenten PC dan spant de trainingsfase decennia. Helaas laat dit ook inzien waarom de ground-up methode buiten de visie van de proef valt.

## Modellen Vergelijken

Om het beste model te kunnen kiezen voor deze proef moeten we deze objectief met elkaar kunnen vergelijken. Dit is het doel van gestandaardiseerde benchmarks. Buiten dat men nu modellen met elkaar kan vergelijken kan men ook zien of er werkelijk progressie wordt gemaakt in de wereld van AI. Sommige benchmarks testen of AI middelbare schoolvragen kan beantwoorden en andere of het kan coderen. Ze functioneren allemaal hetzelfde op het einde: de test geeft de invoer aan het model en evalueert het antwoord.

Het antwoord evalueren en een objectieve score geven is een lastig gegeven. De makkelijkste aanpak is dat de test een meerkeuze bevraging is. Hier kan een programma snel en consequent een score uitdelen. Helaas kunnen niet alle testdomeinen werken met Multiple Choice. Sommige domeinen eisen dat kennis getest via open vragen. De manier om deze dan een score te geven is vaak via de BLEU score. De BLEU score geeft de gelijkenis van een antwoord met een voorbeeld antwoord. Maar sommige vragen kunnen correct beantwoord zijn maar niet lijken op een voorbeeld antwoord. In die gevallen met een mens met expertise de antwoorden evalueren wat een tijdsintensieve opdracht is. Steeds vaker wordt hier zelfs een chatbot gebruikt om het antwoord te evalueren.

De manier waarop vragen gesteld worden is ook zeer belangrijk. We kunnen hier al een eerste onderscheidt maken tussen zero- en few-shot. Deze term gaat erover of het model opgeloste voorbeeldvragen krijgt. Vaak is de few-shot score hoger omdat het betere voorspellingen kan maken als meer data gegeven is. Toch zero-shot een waardige methode omdat het vaak nauwer bij het gebruiksdoeleinde ligt.

Een laatste verbetering aan een benchmark gaat over Chain-of-Thought (CoT). Dit concept is vooral toepasselijk voor Multiple Choice benchmarks want hier vraagt men aan het model om eerst de redenering te geven en dan pas het antwoord. CoT vragen bevordert de scores vaak significant en daarmee dus ook de kwaliteit van het antwoord. Daarbovenop krijgen we meer inzicht in de fouten zelf en kunnen we daarna doelgerichter te werk gaan om het model te verbeteren.

**Stukje over selecteren van HellaSwag en WinoGrande**

Nu het domein van de benchmarks duidelijk is, kunnen we nu benchmarks selecteren die vaak gebruikt worden. Dit is essentieel zodat we zoveel datapunten hebben om een correct startpunt te bepalen. Daarom heb ik mij beperkt tot de HuggingFace OpenLLM leaderboard benchmarks die met taal te maken hebben: HellaSwag en Winogrande.

De HellaSwag benchmark test of een model een gegeven sequentie logisch kan vervolgen. Een voorbeeld uit de dataset is: “The woman sits at the piano and” waarna het model de keuze heeft tussen “she puts her hands on the keys” en “she starts dancing”. De benchmark is specifiek zo dat deze simpel is voor mensen maar moeilijk statistisch te benaderen is. Dat wil zeggen dat een model enkel een hoge score kan behalen als het een intrinsiek inzicht heeft in de scenario’s.

Winogrande test hoe goed een model verwijzing beheerd. Een voorbeeld kan zijn “De Papa is moe want de Zoon heeft niet geslapen, hij was de hele nacht wakker van de honger”, naar wie verwijst “hij”. Ook deze test is makkelijk voor mensen en moeilijk voor LLM’s. Het is een ware test voor de attentie mechanismes die aan de kern van de transformators ligt.