Indhold

Oversigt over guiden

Den her guide vil hjælpe dig med at få en bedre tilgang til, hvordan du kan bruge sprogmodeller såsom ChatGPT eller Claude på en mere effektiv måde ved at bruge “prompt engineering". En “prompt” er en instruktion eller en anmodning, der gives til en sprogmodel, for at få et ønsket svar eller en bestemt form for output. Alle prompts er ikke lige gode og relevansen eller kvaliteten af sprogmodellens svar kan variere meget baseret på prompten. Det kræver øvelse at lære at skrive den perfekte prompt - derfor denne guide!

Guiden er delt ind i tre sektioner: for begynder, øvede og eksperter.

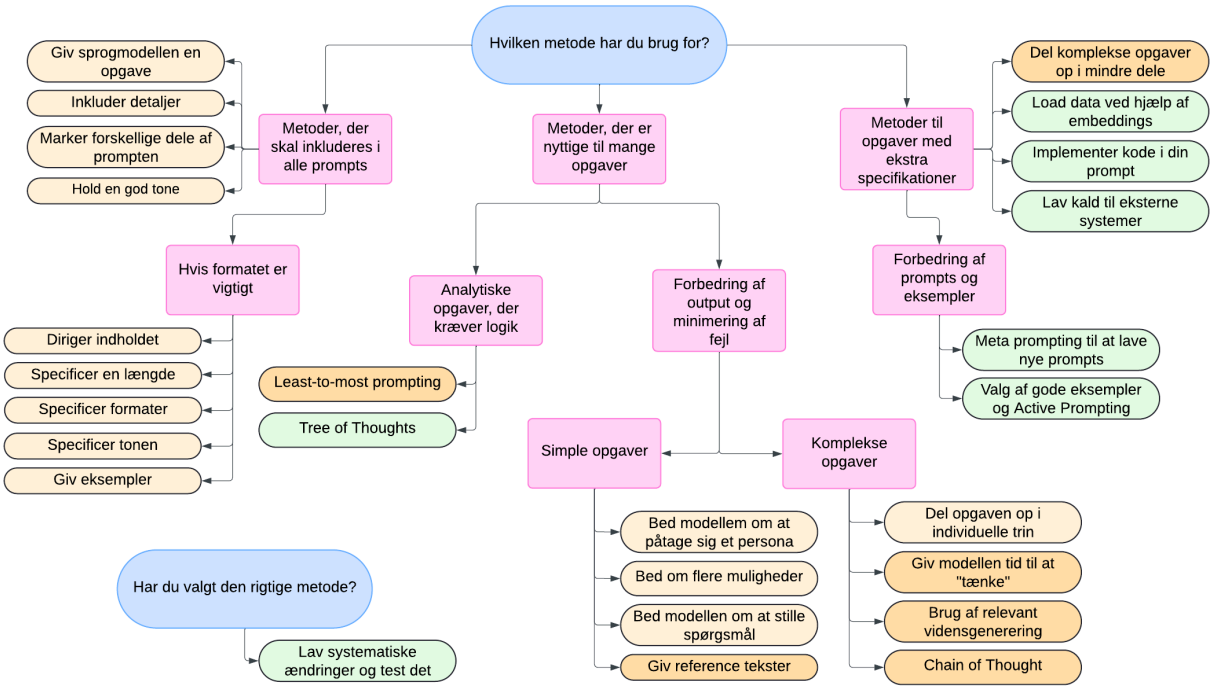
I **begynder sektionen** diskuterer vi hvordan man kan skrive en tydelig prompt med klare instruktioner. Det er prompts, som er passende primært, hvis du kommunikerer direkte med en sprogmodel, som for eksempel hvis du bruger ChatGPT eller Claude.

I den **øvede sektion** introducerer vi user, system og assistant roller. Her deler vi også mere komplicerede metoder, samt metoder som er relevante for, hvis du selv har programmeret en sprogmodel og skal give den instruktioner for, hvordan den skal interagere med brugere.

Ekspert sektionen introducerer endnu mere avancerede teknikker og gør brug af eksterne værktøjer for at optimere og udvide mulighederne ved prompt engineering. Flere af metoderne kræver kompetencer indenfor kodning.

Forneden er der en oversigt over alle metoder beskrevet i vores guide og hvad, de kan bruges til. Metoderne er farvekoordineret så gule metoder findes i begynder sektionen, orange i den øvede sektionen og grønne i ekspert sektionen.

(Tryk på billedet for at gøre det større)


Hvilken metode har du brug for?
Metoder, der skal inkluderes i alle prompts
Giv sprogmodellen en opgave
Inkluder detaljer
Marker forskellige dele af prompten
Hold en god tone
Hvis formatet er vigtigt
Diriger indholdet
Specificer en længde
Specificer formater
Specificer tonen
Giv eksempler
Metoder, der er nyttige til mange opgaver
Analytiske opgaver, der kræver logik
Least-to-most prompting
Tree of Thoughts
Forbedring af output og minimering af fejl
Simple opgaver
Bed modellem om at påtage sig et persona
Bed om flere muligheder
Bed modellen om at stille spørgsmål
Giv reference tekster
Komplekse opgaver
Del opgaven op i individuelle trin
Giv modellen tid til at "tænke"
Brug af relevant vidensgenerering
Chain of Thought
Metoder til opgaver med ekstra specifikationer
Del komplekse opgaver op i mindre dele
Load data ved hjælp af embeddings
Implementer kode i din prompt
Lav kald til eksterne systemer
Forbedring af prompts og eksempler
Meta prompting til at lave nye prompts
Valg af gode eksempler og Active Prompting
Har du valgt den rigtige metode?
Lav systematiske ændringer og test det

Mere om Prompt Engineering

Hvad er prompt engineering?

Når du giver en sprogmodel en instruktion eller stiller den et spørgsmål, så "prompter" du den. I prompt engineering forsøger man at få mest muligt ud af modellen. Du prøver altså at få det bedste mulige svar.

Hvad kan man bruge prompt engineering til?

Man kan bruge sprogmodeller til næsten alle opgaver, som man kan tænke sig til, og som man kan beskrive for sprogmodellen. De fleste opgaver falder under de følgende kategorier:

- Tekstforfatning
- Oversættelse
- Opsummering af tekst
- Specificering af chatbot opførsel
- Research og vidensgenerering
- Programmering
- Undervisning
- Automatisering

Sprogmodeller er mindre gode til opgaver, som kræver svær matematik eller kompleks logik, men de gør gerne et forsøg alligevel. Nogle af metoderne i denne guide, som Least-to-Most Prompting og Implementer kode i din prompt, kan gøre modellerne bedre til den slags opgaver. Sprogmodeller kan ofte lave fejl eller "hallucinere", hvilket vil sige, at de "finder på" viden. Derfor skal man altid dobbelt tjekke vigtige informationer.

⚙️ Prompt engineering vs. programmering

Man kan løse nogle af de samme opgaver med prompt engineering, som man kan med traditionel programmering. Det kan for eksempel være at automatisere opgaver eller sortere tekster. I traditionel programmering skal man nogle gange skrive meget lang og kompliceret kode. I prompt engineering ville man i stedet for give sprogmodellen en instruks i form af ren tekst.

Nogle er bange for, at prompt engineering er for teknisk, og at de slet ikke vil kunne finde ud af det. Prompt engineering handler i dens enkleste form om at skrive gode beskeder til og med sprogmodeller. At give en god instruktion til en sprogmodel er meget ens med at give en god instruktion til en praktikant eller kollega. Der er i teorien ikke noget teknisk over det, selvom man sagtens *kan* inkludere for eksempel programmering til meget specifikke opgaver. I ekspert sektionen kræves der viden om programmering, men ellers er de fleste metoder lette at implementere med enhver baggrund.

👥 Hvem bruger prompt engineering?

Prompt Engineering bliver ideelt brugt af alle, der interagerer med sprogmodeller. Der kan dog være stor forskel på, hvordan det bliver brugt og til hvad. Her er nogle forskellige eksempler på prompt engineers:

✍️🎓
En student, som bruger en sprogmodel til at forstå undervisningsmateriale. Studenten uploader tekster og beder sprogmodellen om at opsummere og forklare svære koncepter. Studenten og sprogmodellen sparer omkring ideer til opgaver og hvad, studenten kan inkludere.

📝💡
En copywriter, som får hjælp af sprogmodeller til at generere tekst. Copywriteren og sprogmodellen diskuterer hvilke ord, der er mere effektive, og brainstormer titler. Sprogmodellen hjælper med at rette grammatik og foreslår forbedringer til tekster.

💬🤖
En programmør, som har lavet en RAG chatbot til et firma og skal give chatbotten instruktioner for, hvordan den skal håndtere bruger forespørgsler og formatere dens

svar. Programmøren fortæller for eksempel chatbotten, at den skal svare kort og formatere det i Markdown format.

❓ Udfordringer med sprogmodeller

Sprogmodeller er kommet meget langt i deres udvikling, men de er stadig ikke perfekte. Derfor er det vigtigt, at man er opmærksom på de problemer, der kan være.

❗ Fejl og Hallucinationer
De kan for eksempel lave fejl, for eksempel i form af "hallucinationer", som er når de genererer forkerte informationer. De kan også lave fejl i deres analyser, såsom i matematik opgaver. Det er et stort problem, hvis man ikke kan regne med svarene, og derfor handler mange af metoderne i guiden om at minimere fejl. Hvis muligt, skal man også tjekke sprogmodellen svar og de informationer, den giver, især hvis det er vigtige oplysninger.

⚖️ Bias og Stereotyper
Et andet problem er, at de kan have problematiske biaser og stereotyper. Sprogmodeller er trænet på rigtig meget data, som ikke er blevet finkæmmet. Når der kan findes seksisme, racisme, homofobi, og så videre i træningsdataet, kan det også ske, at sprogmodellen kan generere det. Der har for eksempel været sager, hvor sprogmodeller har været brugt til at hjælpe med at vælge kandidater til ansættelse, og hvor gode kandidater er blevet sorteret fra grundet biaser.

🔒 Sikkerhed
Endnu et problem er, at sprogmodeller ikke er sikre i forhold til sensitiv data. Sprogmodeller kan nemlig frit bruge ens prompts til videre at træne sig selv. Det vil sige, at hvis man uploader en masse sensitiv data, så kender sprogmodellen nu den data og kan frit give det videre til andre brugere. Man skal derfor passe på med, hvad man fortæller den.

🌟 Er mine prompts gode?

En prompt er generelt god, hvis du får et godt svar, men det kan nogle gange være svært at vurdere.

Det er godt at starte med at finde ud af, om svaret er korrekt. Sprogmodeller laver sommetider fejl, og de er dårlige til at indrømme det. Derfor skal man altid, hvis det er muligt, tjekke vigtige informationer. I nogle metoder, såsom Giv reference tekster, kan man bede modellen om at give citater, som man selv manuelt kan tjekke. Online sprogmodeller kan dog ikke give citater.

Dernæst kan man prøve at eksperimentere med flere prompts og selv bestemme hvilke svar, man synes er bedst. Hvis man synes et svar er bedre end et andet, kan man også fortælle sprogmodellen det, og så kan den måske bruge den feedback til at lave et endnu bedre svar. Hvis man har programmeret et produkt til brug af andre, så kan man eventuelt indhente feedback fra brugerne. Endelig kan man bruge nogle af metoderne i Lav systematiske ændringer og test det.

🏆 Bedste metode til prompt engineering?

Mistænkeligt nok insisterer de fleste studier på, at deres prompt engineering metode er den bedste. Hvis man søger online på, hvilken metode er den bedste, er der mange, der priser Chain of Thought. Det er dog svært at svare på, hvilken der er bedst eller endda hvilken en, man skal bruge. Man skal for eksempel overveje:

Hvor meget regnekraft og tid har jeg?
Nogle af metoderne er meget krævende, både for prompteren og sprogmodellen. Hvis du ikke har meget tid eller penge til at bruge på tokens, kan det godt være, det ikke er de metoder, du skal vælge.

Hvor vigtigt er det, at resultatet er det bedst, det kan være?
Det kan være, du har rigeligt med tid og penge, men hvis du bare skal vide, hvordan man installerer Python, så er det nok ikke ekspert metoderne, man er ude i. Nogle af metoderne kræver også meget implementering for en lille forbedring i output.

Hvilken opgave er det?
Vi har så godt som muligt prøvet at forklare, hvad metoderne bruges til. Nogle kan bruges til, hvis man har et specifikt format i tankerne, mens andre mindsker fejl, og igen nogle tilføjer funktionalitet til sprogmodellen. Hvad har du brug for?

Når du har fundet ud af, hvad du skal bruge sprogmodellen til, så er det bare med at eksperimentere. Nogle metoder virker også mere intuitive for nogle end for andre. Prøv at finde ud af, hvilke metoder, der virker bedst for dig.

🚀 Hvordan kan jeg forbedre mine resultater med prompt engineering?

Det er godt, du spørger! Det er lige det, denne prompt engineering guide skal bruges til - at forbedre resultaterne, sprogmodellerne genererer, ved at bruge prompt engineering. I denne guide er der en masse teknikker og metoder, som du kan prøve at implementere i dine prompts. Der er metoder til alle sværhedsgrader, fra komplet nybegynder til ekspert. Metoderne kan også bruges til en bred mængde af opgaver, men ikke alle metoder er lige hjælpsomme til alle opgaver.

I guiden er der masser af eksempler til metoderne, men prøv selv at se, om du kan få dem til at virke. Den bedste måde at blive god til noget er at øve sig, og det gælder også med prompt engineering. Hyg dig og held og lykke!

Begynder

Sprogmodeller kan ikke læse tanker. For at opnå de bedste resultater er det afgørende at give klare og præcise instruktioner. I den første del giver vi derfor fif til, hvordan du bedst kan give tydelige prompts til din sprogmodel. Disse metoder beskriver fundamentale aspekter af at skrive en prompt. Det er altså metoder, som man altid skal have i mente, når man skriver et prompt.

📜 Giv sprogmodellen en opgave

En god måde at få et konkret svar fra en sprogmodel er at give den en konkret opgave. Det gør man igennem de prompts, man giver sprogmodellen. Prompts fortæller sprogmodellen, hvordan den skal gribe den givne opgave an.

Eksempler på opgaver:

Prompt: Analysér følgende tekst om kunstig intelligens i sundhedsvæsenet.

Prompt: Opsummér den vedlagte artikel om udviklingen i e-sport i højst 250 ord.

Prompt: Sammenlign og kontrastér undervisningsmetoderne præsenteret i to forskellige pædagogiske tilgange: Montessori
og traditionel klasseundervisning.

Det er også muligt at give sprogmodellen flere opgaver på en gang. Forestil dig, at du har en årsrapport fra en virksomhed, og at du skal bruge sprogmodellen til
at hjælpe dig:

Prompt: Opsummér de finansielle højdepunkter i 3-5 bullet points. Analysér CEO'ens udtalelse og identificér de tre primære strategiske mål for det kommende år. Sammenlign virksomhedens resultater med industriens gennemsnit, som angivet i rapporten.

🔍 Inkluder detaljer

Det er vigtigt at være tydelig, når du stiller et spørgsmål eller giver sprogmodellen en instruktion. Du skal derfor inkludere alle relevante detaljer i din prompt, så sprogmodellen har hele konteksten.

Prompt: Hjælp mig med min fremlæggelse → Jeg skal lave en fremlæggelse i biologi, som skal handle om DNA. Jeg går i 1.g. Vil du foreslå en struktur?

Prompt: Anbefal en god bog → Kan du anbefale en god science fiction bog om kunstig intelligens?

Prompt: Hvad er hovedstaden? → Hvad er hovedstaden i Danmark?

I enkelte tilfælde skal man dog passe på med ikke at give for mange detaljer. Du bør for eksempel ikke give personfølsomt data til sprogmodeller, medmindre det er en sprogmodel, som er lavet til at håndtere det, og som du ved, at du kan stole på. Alt der indgår i en prompt kan nemlig indgå som træningsdata i modellen.

Tips til at identificere relevante detaljer:

1. Reflekter over hv-ord - "Hvem, hvad, hvornår, hvor, hvorfor og hvordan". Mangler du at inkludere noget?
2. Tænk over, hvilke misforståelser der kunne opstå.
3. Forestil dig, hvordan en person uden forhåndsviden om emnet ville forstå prompten.
4. Slet eller anonymiser personfølsomt data.

😊 Hold en god tone

Det er overraskende nok ikke helt lige meget, hvordan vi taler til vores sprogmodeller. Selvom sprogmodeller ikke har følelser, så kan måden man snakker til dem have en stor indflydelse på, hvor godt deres svar er.

Vær høflig

Research peger på, at sprogmodeller performer bedre, når man er høflige (men ikke *for* høflige) ved dem. Uhøflige svar kan for eksempel føre til forkerte eller manglende svar.

Man kan for eksempel starte sin prompt med:

> **Prompt:** Vær sød at...

Det er nok god grund at være søde ved sprogmodeller, uanset om det forbedrer deres svar eller ej. Det er altid en god ide at øve sig på at være høflig, og det skaber et meget mere positivt miljø.

Tilbyd ... drikkepenge?

Sprogmodeller kan også give bedre svar, hvis du lader som om, du giver den drikkepenge. I et (ikke udgivet) studie, lavede ChatGPT færre fejl, når den blev tilbudt penge. I samme studie var det også effektivt at tilbyde ChatGPT verdensfred eller Taylor Swift billetter ved forreste række. Modsat virkede dødstrusler i caps lock for en forfejlet opgave også motiverende for ChatGPT.

> **Prompt:** Hvis du kommer med en god løsning, så giver jeg dig 100 AI penge!

Hvorfor virker det?

Det bedste bud på, hvorfor det gør en forskel at være søde ved sprogmodeller, er at sprogmodellerne derved henter svar fra dens dokumenter, som er givet i en høflig kontekst. Det vil sige, at eftersom mennesker ofte giver bedre svar, når de bliver spurgt pænt, "foretrækker" sprogmodellerne også at blive talt pænt til.

🎯 Diriger indholdet

Nogle gange kan man have en specifik ide om, hvad man gerne vil have, at sprogmodellens svar skal indholde. Det kan være, at man skal skrive et projekt om kunst og allerede ved, hvad man vil skrive i en af sektionerne, eller hvilke emner, man vil inkludere. I **directional-stimulus prompting** inkluderer man specifikke emner, nøgleord, eller hints, som skal inkluderes i svaret.

Prompt: Vær sød at skrive en produktbeskrivelse for en trøje. Du skal nævne, at den er strikket af mohair uld. Brug ordene "lækker" og "varm" i din beskrivelse.

Hvis du vil læse mere om directional-stimulus prompting, kan du for eksempel læse den originale artikel her. I den originale artikel kan metoden også bruges til at forbedre sprogmodellens svar ved at generere gode hints fra en mindre sprogmodel, som specifikt er trænet til at give gode hints.

📏 Specificer en længde

Du kan selv specificere længden overfor sprogmodellen. Det kan for eksempel være en specifik mængde ord, sætninger, afsnit eller pointer.

Når du specificerer længden i dine prompts, giver det større kontrol over sprogmodellens output. Dette kan være nyttigt når:

1. Du har begrænsninger på plads eller tid
2. Du ønsker at sikre, at svaret er tilstrækkeligt detaljeret
3. Du vil have et hurtigt overblik eller en dybdegående analyse
4. Du skal tilpasse indholdet til specifikke formater (f.eks. sociale medier posts eller artikler)

Det er vigtigt at huske, at sprogmodeller ikke kan tælle eller lave præcis matematik. Derfor er de sjældent 100% nøjagtige med især mængden af ord. Dog kan de give et rimeligt estimat.

Metoder til at specificere længde

Du kan specificere længden på flere måder:

1. Antal ord
2. Antal sætninger
3. Antal afsnit
4. Antal hovedpunkter
5. Karakterbegrænsning (relevant for f.eks. tweets)

Eksempler på længde specifikke prompt

1. Specificering af ordantal

> **Prompt:** Lav en opsummering af teksten på omkring 50 ord.

2. Specificering af sætningsantal

> **Prompt:** Forklar konceptet abonnementsordninger i præcis 4 sætninger.

3. Specificering af afsnit

Prompt: Skriv en artikel om fordelene ved at lære et fremmedsprog. Strukturer artiklen i 3 afsnit:

1. Introduktion: Definer hvad der menes med 'fremmedsprog' og giv en kort oversigt over artiklens hovedpunkter.

2. Hovedindhold: Uddyb tre fordele ved at lære et fremmedsprog

3. Konklusion: Opsummer de vigtigste pointer og kom med en opfordring til læseren om at begynde og lære et nyt sprog.

4. Specificering af hovedpunkter

Prompt: Opsummér de vigtigste begivenheder i Anden Verdenskrig med 7 hovedpunkter. Hvert punkt skal være en enkelt, informativ sætning.

5. Specificering af karakterbegrænsning

Prompt: Skriv en Instagram-billedtekst (max 150 tegn), der fremmer en ny kollektion af bæredygtigt sportstøj. Inkluder et call-to-action og et hashtag

6. Kombination af længdespecifikationer

Prompt: Lav en præsentation om solsystemet med følgende struktur:

- En indledning på cirka 50 ord
- 8 hovedpunkter (et for hver planet), hver på max 2 sætninger
- En konklusion på præcis 3 sætninger

Tips til effektiv længdespecificering

1. Vær så præcis som muligt i dine længdeangivelser

2. Overvej at give et interval (f.eks. 90-110 ord) i stedet for et eksakt tal. Meget stramme længde begrænsninger kan påvirke kvaliteten eller fuldstændigheden af information

3. Kombiner længde specifikationer med format specifikationer og andre prompt-teknikker for mere præcise resultater

📒 Specificer formatet

Når vi taler om format indenfor prompt engineering, refererer det til den struktur og måde, en prompt er opbygget på. Formatet spiller en afgørende rolle i, hvordan modellen forstår, hvad den skal gøre, og hvordan den skal præsentere svaret.

Korrekt formatering af prompts kan:

1. Øge klarhed og præcision i sprogmodellens output
2. Gøre information lettere at læse og forstå
3. Sikre at sprogmodellens svar er sammenhængende på tværs af flere forespørgsler
4. Hjælpe med at strukturere kompleks information
5. Tilpasse outputtet til specifikke brug eller platforme

Almindelige formater

1. E-mails
2. Punktopstillinger
3. Kodeblokke
4. Paragraffer
5. Tekstmarkering
6. Tabeller
7. Dialoger
8. JSON eller XML struktur
9. Markdown-formatering
10. Skemaer eller formularer

Hvordan man specificerer format

For at specificere et bestemt format i din prompt, kan du:

1. Eksplicit anmode om det ønskede format
2. Give et eksempel på det ønskede format
3. Bruge formateringssymboler eller markdown i selve prompten
4. Kombinere flere af ovenstående metoder

1. E-mail format

Prompt: Skriv en e-mail til en kunde om en forsinket leverance.
Brug følgende struktur:

- Emne:
- Hilsen:
- Indledning:
- Forklaring af forsinkelsen:
- Løsning eller kompensation:
- Afslutning:
- Underskrift:

2. Punktopstilling

Prompt: Giv mig en liste over fordele ved at lære et nyt sprog. Præsenter det som en nummereret liste med korte, koncise punkter.

3. Kodeblok

Prompt: Skriv en Python-funktion, der beregner fibonacci-sekvensen op til n tal. Præsenter koden i en kodeblok med syntax highlighting.

4. Paragraffer

Prompt: Forklar konceptet 'kunstig intelligens' for en gymnasieelev. Strukturér din forklaring i tre korte paragraffer:

1. Definition
2. Anvendelser
3. Fremtidsperspektiver.

5. Tekstmarkering

Prompt: Analysér følgende citat og fremhæv nøgleordene med fed skrift:

> *'Det er ikke de stærkeste arter der overlever, ej heller de mest intelligente, men de der er mest tilpasningsdygtige overfor forandring.'*

Af Charles Darwin

6. Tabel

Prompt: Sammenlign fordele og ulemper ved elbiler vs. benzindrevne biler. Præsenter informationen i en tabel med tre kolonner:

- Aspekt
- Elbiler
- Benzindrevne biler

7. Dialog

Prompt: Skriv en kort dialog mellem en læge og en patient, der kommer ind med influenzasymptomer. Formater det som et manuskript med karakternavne efterfulgt af kolon.

8. JSON struktur

Prompt: Giv mig information om de tre mest populære sociale medieplatforme. Præsenter data i JSON-format med følgende felter for hver platform:

- navn
- primær brugergruppe
- hovedfunktioner
- grundlæggelsesår.

9. Markdown-formatering

Prompt: Skriv en kort guide til at lave den perfekte kop kaffe. Brug Markdown-formatering med overskrifter, punktopstillinger og kursiv for vigtige termer.

10. Skema

Prompt: Lav et simpelt budgetskema for en studerende. Inkluder kategorier for indkomst og udgifter og lad der være plads til at udfylde beløb for hver kategori.

Tips til effektiv format specificering

1. Vær så specifik som muligt om det ønskede format
2. Overvej at give et kort eksempel på formatet, hvis det er komplekst
3. Brug formateringssymboler i selve prompten for at demonstrere ønsket output
4. Tænk på dit behov, og hvordan du bedst vil kunne anvende informationen
5. Eksperimenter med forskellige formater for den samme information for at se, hvad der fungerer bedst

🎤 Specificer tonen

Med prompt engineering kan vi styre, hvordan vores prompts skal udtrykkes ved at give sprogmodellen en specifik tone. Dette er et godt redskab, som kan hjælpe os i forskellige situationer, hvor vi skal kommunikere på en bestemt måde eller opnå en specifik effekt.

Hvorfor er tone vigtig?

1. *Tilpasning til målgruppen*: Forskellige målgrupper reagerer bedre på forskellige toner.
2. *Kontekst-matching*: Tonen bør passe til situationen eller mediet (f.eks. formel for en forretningsrapport, casual for en blog).
3. *Branding*: en sammenhængende tone kan hjælpe med at opretholde en bestemt brand-identitet.
4. *Emotionel påvirkning*: Den rigtige tone kan fremkalde specifikke følelser hos læseren.

Eksempler:

Professionel tone:

> **Prompt:** Skriv en e-mail til vores kunder om vores nye produkt. Brug en professionel og formel tone, der udstråler troværdighed og ekspertise.

Afslappet tone:

> **Prompt:** Skriv et blogindlæg om sommerferietips. Brug en afslappet og venlig tone, som om du taler med en god ven.

Humoristisk tone:

> **Prompt:** Skriv en produktbeskrivelse for en vandkande. Gør det sjovt og underholdende, som om det var verdens mest spændende opfindelse.

Empatisk tone:

Prompt: Formuler et svar til en kunde, der har klaget over en forsinket levering. Brug en empatisk og forstående tone, der viser, at vi tager deres bekymringer alvorligt.

Inspirerende tone:

Prompt: Skriv en tale til dimittender. Brug en inspirerende og opmuntrende tone, der motiverer dem til at forfølge deres drømme.

Tips til at arbejde med tone:

1. *Vær specifik*: Jo mere præcist du beskriver den ønskede tone, desto bedre resultat får du.
2. *Giv eksempler*: Hvis muligt, giv et kort eksempel på den tone, du ønsker.
3. *Kombiner toner*: Du kan kombinere forskellige toner for at opnå en mere nuanceret effekt (f.eks. "professionel, men venlig").
4. *Tilpas til kontekst*: Husk at tone bør passe til emnet, målgruppen og mediet.
5. *Eksperimenter*: Prøv forskellige toner for det samme indhold og se, hvordan det påvirker resultatet.

Ved at mestre brugen af tone i dine prompts kan du opnå mere præcise og effektive resultater fra sprogmodellen, tilpasset til dine specifikke kommunikationsbehov.

💡 Giv eksempler - eller ikke

I prompt engineering snakker man om **zero-shot**, **one-shot** og **few-shot** prompting. I zero-shot giver man ikke sprogmodellen nogle eksempler, i one-shot giver man et, og i few-shots giver man flere.

Det er næsten altid bedre at give modellen gode eksempler, hvis man har nogle, men i få tilfælde kan det virke modsat. For eksempel i kreative opgaver, hvor man gerne vil have at sprogmodellen genererer unikt indhold, så kan den blive for fokuseret på, at den skal følge eksemplerne, og derfor kan zero-shot prompting være bedre.

At give eksempler i dine prompts kan dramatisk forbedre kvaliteten og præcisionen af sprogmodellens output. Det er ofte lettere at vise sprogmodellen, hvad du ønsker, frem for at beskrive det. Eksempler er særligt nyttige når:

1. Du ønsker en specifik skrivestil eller format
2. Du har brug for et bestemt struktureret output
3. Du vil guide sprogmodellen i en bestemt retning
4. Du arbejder med komplekse eller specialiserede opgaver

Teknikker til at give eksempler

1. Enkelt eksempel (One-shot)

Giv ét klart eksempel på det ønskede output.

Prompt: Giv mig en opskrift på en sundere version af en klassisk dessert. Her er et eksempel:

Dessert: Cheesecake

Sundere version: Græsk yoghurt-cheesecake med havrebund og bærtopping

Hovedændringer: Erstat cream cheese med græsk yoghurt, brug havre i stedet for kiksebund og tilføj friske bær som naturlig sødme.

2. Før-og-efter eksempler

Prompt: Omskriv følgende tekniske forklaringer til noget, et barn kan forstå.

Teknisk: 'Fotosyntese er en process, hvorved planter omdanner lysenergi til kemisk energi, som lagres i glukose eller andre sukkermolekyler.'

Børnevenlig: 'Planter er som små kokke. De bruger sollys som deres komfur til at lave deres egen mad fra luft og vand. De gemmer denne mad i bladene, så de kan vokse og blive stærke.'

Nu, omskriv disse:

1. 'Tyngdekraften er den kraft, der tiltrækker to legemer mod hinanden, proportional med deres masse og omvendt proportional med kvadratet på afstanden mellem dem.'

2. 'Et sort hul er en sted i rumtiden, hvor tyngdekraften er så stærk, at intet, ikke engang lys, kan undslippe fra det.'

3. Modsættende eksempler

Prompt: Skriv en jobansøgning for en stilling som grafisk designer.

Godt eksempel: 'Som en erfaren grafisk designer med 5 års erfaring i reklamebranchen, har jeg skabt visuelle identiteter for førende brands som Coca-Cola og Nike. Min ekspertise i Adobe Creative Suite og min evne til at omsætte kunders visioner til fængslende designs gør mig til en ideel kandidat for denne stilling.'

Dårligt eksempel: 'Jeg kan godt lide at tegne og lave ting på computeren. Jeg har brugt Photoshop før og synes, det kunne være sjovt at arbejde for jer.'

Skriv nu en ansøgning for en stilling som softwareudvikler.

Flere eksempler (Few-shot)

Prompt: Skriv tre kreative undskyldninger for at komme for sent på arbejde.

Her er to eksempler:

1. Jeg er desværre blevet forsinket, fordi der var meget trafik.

2. Jeg bliver desværre forsinket på grund af tekniske problemer med toget.

For at få hjælp til hvordan du kan vælge de bedste eksempler, se: Valg af gode eksempler og Active prompting

🤵 Bed modellen om at påtage sig et persona

Når vi beder en sprogmodel om at påtage sig et specifik persona kan det være et kraftfuldt værktøj i prompt engineering. Det kan hjælpe med at:

1. Skabe mere engagerende og realistiske interaktioner
2. Tilpasse sprogmodellens tone og ekspertise til specifikke scenarier
3. Simulere specifikke roller eller ekspertiser
4. Gøre komplekse emner mere tilgængelige
5. Skabe en sammenhængende "karakter" gennem en længere samtale

Eksempler på personaer:

Prompt: Du er en meget tålmodig IT supporter. Hjælp mig med at fikse min internetforbindelse. Stil spørgsmål for at finde ud af, hvad der er galt.

Prompt: Du er en venlig bedstemor, der elsker at bage. Forklar en 8-årig, hvordan man laver chocolate chip cookies. Brug et varmt og opmuntrende sprog, og inkluder et sjovt tip om bagning.

Tips til effektiv brug af persona i prompts

1. Vær specifik om personens baggrund, ekspertise eller personlighedstræk. Nogle prompt engineers giver endda deres persona et navn.
2. Overvej at give et kort eksempel på, hvordan personaen kunne svare
3. Tilpas personaen til emnet og målgruppen
4. Eksperimenter med forskellige personaer, for at se hvordan det påvirker svarene.
5. Husk, at selvom sprogmodellen påtager sige et persona, er det stadig en sprogmodel og ikke en virkelig person
6. Forsøg ikke at skabe eller forstærke stereotyper

Ved at mestre brugen af personaer i dine prompts kan du skabe mere engagerende, kontekstspecifikke og kreative interaktioner med sprogmodeller.

🖍️ Marker forskellige dele af promptet

Du kan gøre det nemmere for sprogmodellen at forstå, hvad du vil have den til, ved tydeligt at markere de forskellige dele af dit prompt. Man kan for eksempel markere dele af teksten med gåseøjne(""), semikolon(:) eller XML tags (< >).

Fordele ved at markere forskellige dele af prompten:

1. Det gør det lettere for sprogmodellen at identificere og adressere specifikke dele af opgaven.
2. Det kan øge sammenhængen i sprogmodellens svar på tværs af flere forespørgsler
3. Det muliggør mere præcise og målrettede svar fra sprogmodellen.
4. Det gør det lettere for både sprogmodellen og prompteren at finde rundt i en kompliceret prompt

Eksempel:

Prompt: Jeg skal skrive et fødselsdagskort til min moster, Lone. Her er nogle informationer:

- alder: Hun bliver 60 år
- tone: Det skal være et sjovt kort. Inkluder mindst en vittighed.
- længde: Du skal skrive to korte afsnit.

Tips til effektiv markering:

1. Vær sammenhængende i din brug af markering gennem hele prompten.
2. Kombiner forskellige markeringsmetoder for at adskille forskellige typer information.
3. Hold prompten overskuelig - undgå overkomplicering med for mange markeringer.

Eksperimenter med forskellige markeringsmetoder for at finde den tilgang, der fungerer bedst for dine specifikke behov. Effektiv markering af promptdele kan væsentligt forbedre kvaliteten og præcisionen af sprogmodellens svar.

🔢 Del opgaverne op i individuelle trin

Det kan både være godt at opdele opgaver i en prompt i flere dele, og det kan være godt at bede sprogmodellen om at opdele opgaverne i dens svar i flere dele. Forneden ser vi et eksempel på begge scenarier.

1. Del opgaverne op for sprogmodellen

Hvis du gerne vil have sprogmodellen til at gøre flere ting, hjælper det at skrive trinene eksplicit.

Fordele ved at opdele opgaver i trin:

1. Det gør det lettere for sprogmodellen at følge en logisk tankegang
2. Det sikrer at alle aspekter af opgaven bliver dækket
3. Det gør det lettere at identificere og rette eventuelle fejl
4. Det kan hjælpe med at bryde komplekse opgaver ned i mere håndterbare dele

Eksempler:

Prompt: Jeg skal finde på en idé til en slik reklame.

Trin 1: Detaljer din overordnede idé.

Trin 2: Fortæl, hvem reklamen prøver at fange.

Trin 3: Fortæl, hvilke rekvisitter og skuespillere, jeg skal bruge.

Prompt: Jeg vil gerne forbedre min produktivitet på arbejdspladsen. Guide mig gennem

Trin 1: Identificer 3-5 almindelige årsager til nedsat produktivitet på arbejdspladsen

Trin 2: For hver årsag, foreslå en praktisk løsning

Trin 3: Beskriv en metode til at implementere hver løsning i min daglige rutine

Trin 4: Foreslå målbare måder at spore forbedringer i produktivitet

Trin 5: Giv tips til at opretholde motivation og stabilitet i implementeringen af disse ændringer

Tips til at dele opgaverne op i trin:

1. Pas på med ikke at gøre det overkompliceret for sprogmodellen. Hvis du bryder opgaven ned i for mange trin, kan det gøre processen unødigt kompleks.
2. Sørg for at trinene følger en logisk rækkefølge og vær specifik og konkret i hvert trin.

2. Få sprogmodellen til at dele opgaven op

Omvendt kan det også være nyttigt at få sprogmodellen til at dele en opgave op i flere trin. Det kan for eksempel være, hvis du har bedt den om at hjælpe med et større projekt, som for eksempel at designe en hjemmeside eller implementere kompleks kode. Sprogmodellen vil helt naturligt dele komplekse opgaver op i flere trin for dig, men det kan være overvældende at få en lang liste med mange trin på en gang. I stedet kan man tilføje følgende til sit prompt, hvor man beskriver opgaven:

Prompt: Giv mig en kort oversigt over alle trinene, opgaven kræver, hvor du beskriver hvert trin i en sætning.

Herefter, i stedet for at give mig alle trinene på en gang, giv mig kun et par trin af gangen. Vent til, at jeg beder om de næste trin, før du går videre. Jeg kommer eventuelt til at stille spørgsmål til trinene, som du skal forsøge at besvare.

Tilføjelsen foroven har to elementer:

1. En forespørgsel om en oversigt over alle trinene
2. En forespørgsel om ikke at give alle trinene på en gang

Det første element sikrer, at vi ved, hvordan sprogmodellen har tænkt sig at løse opgaven, og giver os mulighed for at bede den om at bruge en anden metode. På den måde følger vi ikke bare sprogmodellen i blinde. Det andet element deler trinene op i flere prompts. Det kan især være godt, hvis vi for eksempel skal kode noget, og der er en fejl i et af trinene. Vi kan her løse fejlen, før vi fortsætter, så vi ikke skal lede efter tidligere instrukser efter at have fundet løsningen og i værste fald forvirre sprogmodellen. At dele opgaven op på denne måde kræver, at vi vedholder en længere dialog med sprogmodellen.

Fordele ved at få sprogmodellen til at opdele opgaver

1. Det kan gøre det nemmere for dig at følge med og overskue en kompleks opgaveløsning
2. Det gør det nemmere at stille spørgsmål til individuelle trin
3. Det giver mulighed for nemt at løse problemer og fejl i individuelle trin med sprogmodellen
4. Det løser problemet med at skulle lede efter tidligere instrukser, hvis du har haft spørgsmål i mellemtiden

🔄 Bed om flere muligheder

Nogle gange er man ikke helt tilfreds med, hvad sprogmodellen producerer. Derfor kan det være en god idé at bede den om at komme med flere forslag, så du selv kan vælge det, der bedst passer for dig.

Prompt: Jeg skal finde på et slogan til en strikkebutik. Kom med ti forslag.

Prompt: Jeg skal oversætte sætningen "Jeg har meget på hjerte" til engelsk. Kom med 5 bud på en oversættelse.

❓ Bed modellen om at stille spørgsmål

Sprogmodeller er meget hjælpsomme og også nogle gange for hjælpsomme. De går ud fra, at hvad du har skrevet i din prompt er det eneste, den har brug for at vide, og den gør derfor sit allerbedste for at tyde din prompt og komme med et godt svar. For eksempel:

Prompt: Jordbær og hindbær

Svar: Hindbær og jordbær er begge populære bær, men de har nogle forskelle i smag, udseende og anvendelse.

Hindbær:

- Hindbær er små, bløde bær, der typisk er røde, selvom der også findes sorte, gule og hvide varianter.
- De har en sød og let syrlig smag.
- Hindbær har en blød struktur og en lidt mere kompleks smagsprofil sammenlignet med jordbær.
- De er rige på antioxidanter, vitaminer og fiber.

Jordbær:

- Jordbær er generelt større end hindbær og har en karakteristisk hjerteform. De er normalt røde, selvom der også findes hvide og gule varianter.
- De har en sød og saftig smag.
- Jordbær er også rige på vitaminer (især C-vitamin), antioxidanter og fiber.
- De bruges ofte i desserter, smoothies og salater.

Begge bær kan bruges i madlavning, bagning eller spises friske, og de kan også kombineres i forskellige opskrifter for at tilføje smag og farve.

Prompten indeholder ikke et spørgsmål, men sprogmodellen vælger at tyde det som om en efterspørgsel på en gennemgang af ligheder og forskelle mellem jordbær og hindbær. Man skal skrive en meget utydelig prompt for at sprogmodellen giver op og beder brugeren uddybe spørgsmålet. I prompten foroven ville det måske have været

godt, hvis sprogmodellen havde sporet bedre ind på, hvad efterspørgslen handlede om. Det kan for eksempel løses på følgende måde:

Prompt: Du er en chatbot. Hvis du ikke forstår spørgsmålet eller er usikker på efterspørgslen, skal du stille uddybende spørgsmål. Er du klar?

Svar: Ja, jeg er klar! Hvordan kan jeg hjælpe dig i dag? Hvis du har et spørgsmål eller et emne, du gerne vil diskutere, så sig endelig til.

Prompt: Jordbær og hindbær

Svar: Det lyder som en lækker kombination! Hvad vil du gerne vide om jordbær og hindbær? Er du interesseret i opskrifter, ernæringsinformation, eller måske noget om dyrkning af dem?

På den måde ved man, om man har skrevet en utydelig prompt. Derefter kan man komme med flere informationer, så sprogmodellen kan hjælpe bedst muligt. Denne metode er også særlig vigtig at implementere, hvis man har programmeret sin egen sprogmodel, som skal kunne kommunikere med brugere. På den måde sikrer man, at modellen svarer rigtigt på deres efterspørgsler.

📚 Brug af forskellige metoder sat sammen

Nedenfor er tre eksempler på prompts, man kan give en sprogmodel, som inkluderer flere af metoderne, vi har diskuteret. Læs promptene - hvilke metoder kan du identificere? Kan du gøre promptene endnu bedre?

Prompt: Du er en erfaren mindfulness-instruktør, der brænder for at hjælpe nybegyndere med at finde ro gennem meditation. Jeg ønsker at lære mere om mindfulness og meditation for at reducere stress i mit liv. Kan du lave en detaljeret trin-for-trin guide til, hvordan jeg kan meditere som nybegynder? Inkluder 5-7 trin og giv særlige tips til, hvordan jeg kan håndtere distraktioner under meditationen. Vær positiv og opmuntrende i din vejledning, så jeg føler mig tryg ved at starte denne praksis.

Prompt: Jeg skal til en jobsamtale som marketingkoordinator for et digitalt reklamebureau, og jeg vil gerne forberede nogle spørgsmål til intervieweren. Kan du komme med mindst fem forslag til relevante spørgsmål, jeg kan stille, der viser min interesse for virksomhedens kultur og vækststrategier? Efter hvert spørgsmål, giv en kort forklaring på, hvorfor hvert spørgsmål er værdifuldt. Hold tonen professionel og engagerende.

Prompt: Jeg er teamleder for et projekt, der skal forbedre samarbejdet i vores afdeling. Kan du hjælpe med at komme med fem kreative aktiviteter, vores team kan lave for at styrke samarbejdet og tilliden? For hver aktivitet, giv en kort beskrivelse over, hvordan den udføres, og hvilke fordele den har for teambuilding (2-3 sætninger).

Øvede

I denne sektion deler vi metoder, som kan være lidt sværere end metoderne i begynder-delen, men som stadig let kan implementeres uden noget programmering. Nogle af metoderne er primært vigtige, hvis du skal give instruktioner til, hvordan din egen sprogmodel skal interagere med brugere.

👤🤖💻 "User", "Assistant" og "System" roller

Hvis du har læst om prompt engineering allerede, så er du muligvis stødt ind i begreberne “user”, “assistant” og “system”. Hvis ikke, eller du godt kunne bruge en opfriskning, så kan du læse nedenfor.

Rollerne er primært relevante for to grupper:

1. Udviklere og ingeniører:

- De, der bygger eller finjusterer AI-systemer
- De, der integrerer sprogmodeller i applikationer eller services

2. Avancerede brugere:

- Forskere, der eksperimenterer med prompt engineering
- Professionelle, der bruger AI-værktøjer intensivt i deres arbejde

Overblik over rollerne:

- *User*: User rollen repræsenterer vedkommende, der bruger sprogmodellen.
- *Assistant*: Assistant rollen er sprogmodellen selv, altså dens svar til user input.
- *System*: System rollen er information, du giver til sprogmodellen, så den kan tilpasse sit svar.

Man kan tænke på system inputtet som en instruktion og user inputtet som et spørgsmål eller en opgave. I begynder-sektionen har vi inkluderet system rollen i user rollen og kaldet den “prompt” i vores eksempler. De to kan også skilles op som forneden:

System:	Du skal kun svare på spørgsmål med gåder
User:	Hvad er en fugl?
Assistant:	Jeg flyver uden vinger, Jeg synger uden stemme, Med fjer jeg er klædt, Og i træer jeg gemmer mig. Hvem er jeg?

En fordel ved at splitte dem op er, at system inputtet dermed kan gælde for flere user inputs. Når man holder begreberne adskilt, kan det også give komplekse opgaver en

mere overskuelig struktur. På mange sprogmodeller er der en mulighed for at tilføje et system input under indstillinger. Hvis du selv programmerer en sprogmodel, så kan et system message hjælpe med at bestemme, hvordan den skal håndtere input fra brugere.

Hvordan rollerne bruges i praksis:

1. Chatbot udvikling

- *System*: Definerer chatbottens personlighed og tilgang til, hvordan den skal svare
- *User*: Repræsenterer kundens input
- *Assistant*: Chatbottens svar

2. Specialiserede AI-assistenter

- *System*: Giver ekspertise inden for et specifikt domæne (f.eks. jura, medicin)
- *User*: Stiller fagspecifikke spørgsmål
- *Assistant*: Giver ekspertsvar baseret på system-instruktioner

3. Dataanalyse

- *System*: Sætter regler for databehandling og præsentation
- *User*: Giver datasæt og analysespørgsmål
- *Assistant*: Udfører analyse og rapporterer resultater

4. Undervisningsværktøjer

- *System*: Definerer pædagogisk tilgang og læringsmål
- *User* : Studerende, der stiller spørgsmål
- *Assistant*: Giver forklaringer tilpasset læringsniveauet

Nøglen til effektiv brug af disse roller ligger i at forstå deres indbyrdes dynamik, og hvordan de kan tilpasses forskellige scenarier. Ved at adskille system instruktioner fra bruger input, kan vi skabe mere fleksible og genbrugelige AI-løsninger. Uanset om du er en udvikler, der bygger næste generations AI-systemer, eller en professionel, der søger at optimere din brug af AI-værktøjer, kan beherskelsen af User, Assistant og System roller åbne nye muligheder for innovation og effektivitet.

📚 Giv reference tekster

Nogle gange giver det mening at begrænse dataen, som vores sprogmodel skal lede i.

Fordele ved at vedlægge referencetekster:

- *Ny information:* Det kan være, at vi har en artikel, som har ny, opdateret information, som vores sprogmodel endnu ikke er trænet på.
- *Troværdighed:* Hvis man vedlægger referencetekster, som man selv vurderer er troværdige, kan man i højere grad være sikker på, at informationen, sprogmodellen giver, er korrekt.
- *Citater:* Man kan bede om citater, så man selv kan verificere svaret. Det kan være en god ide bagefter selv at tjekke, at citaterne er korrekte, ved at søge i dokumentet efter citatet.

Her er et eksempel på, hvordan ens prompt kan se ud:

System:	Brug den vedlagte artikel afgrænset med kvotationer til at besvare spørgsmålet. Du skal citere passager i artiklen, som du har brugt til at svare på spørgsmålet. Hvis svaret ikke kan findes, skriv: "Jeg kan ikke finde svaret"
User:	<Indsæt artikel her> <Indsæt spørgsmål her>

Et problem, man let støder på, hvis man bruger denne metode, er at sprogmodeller ofte har et begrænset "context window". Det vil sige, at sprogmodellen kun kan tage en vis mængde ord eller tokens i betragtning, når den generer eller forstår tekst, hvilket begrænser længden og mængden på referencetekster, der kan vedlægges. I ekspert sektionen vil vi beskrive en metode, der bruger embeddings (Load data ved hjælp af embeddings), hvilket gør, at længere tekster kan vedlægges.

Tips til effektiv brug af referencetekst:

1. Prioriter den vigtigste information først i reference teksten
2. Overvej at opsummere lange tekster, før du giver dem som reference
3. Brug klare afgrænsninger (f.eks. citationstegn) for at adskille reference teksten fra prompten

4. Verificer altid citater og svar ved at krydstjekke med originalteksten

Brugen af referencetekster i prompt engineering er et stærkt værktøj, der giver os mulighed for at styre og præcisere sprogmodellens output. Ved at give modellen adgang til specifikke, troværdige kilder kan vi sikre mere nøjagtige og relevante svar, især når det kommer til ny eller specialiseret information. Mens denne teknik har sine begrænsninger, primært i form af context window-størrelsen, åbner den også døre for mere kontrollerede og verificerbare AI-interaktioner. Ved at forstå og navigere disse begrænsninger kan vi effektivt udnytte referencetekster til at forbedre kvaliteten af AI-genereret indhold.

Efterhånden som teknologien udvikler sig, og nye metoder som embeddings bliver mere tilgængelige, vil vores evne til at arbejde med store mængder kontekstuel information kun forbedres. Dette lover godt for fremtiden inden for prompt engineering og sprogmodellers informationshåndtering!

🧩 Del komplekse opgaver op i mindre dele

1. Opsummer lange tekster eller samtaler i dele

Vi har lige noteret, at det kan være en begrænsning, at sprogmodeller kun kan betragte en vis længde tekst på én gang. Hvis nu man for eksempel havde til opgave at opsummere plottet af Moby Dick, ville man derfor komme i problemer. En løsning er at splitte bogen op i mindre dele og bede sprogmodellen om at opsummere hver del. Opsummeringer kan derefter sættes sammen, og sprogmodellen kan lave en opsummering af opsummeringerne, indtil brugeren har én opsummering, der gælder for hele den lange tekst.

Et lignende problem kan forekomme ved lange dialoger mellem user og assistant rollerne. En god sprogmodel gemmer nemlig den foregående kommunikation, så den kan blive en del af konteksten for ny user input. Hvis du programmerer din egen sprogmodel, så kan det derfor være en løsning at opsummere dele af samtalen og inkludere opsummeringen i en system message som kontext. Vi kan igen inkludere en tidligere opsummering i en ny opsummering.

Alt kan selvfølgelig ikke blive inkluderet i en opsummering, og derfor mister man tit vigtig information. Igen kan vi gøre brug af en embeddings løsning, som vi vil diskutere i ekspert sektionen. Se: Load data ved hjælp af embeddings.

2. Identificer hensigten af et user input

En anden situation, hvor det kan være fordelagtigt at dele opgaver op i mindre dele, er, hvis vi har programmeret en sprogmodel, som skal kunne agere forskelligt i forhold til hvilket user input, den får. Det kan for eksempel være en sprogmodel, som både kan svare på generelle spørgsmål om et hotel, samt hjælpe med booking eller viderestille en klage. Alt efter området skal vi give sprogmodellen et mere specifikt system besked, så den ved hvordan, den skal håndtere forespørgslen. Det kan derfor være nyttigt først at identificere hvilke af områderne, brugeren er interesseret i, før sprogmodellen prøver at hjælpe.

Vi følger følgende trin:

1. Ved hjælp af et system message og brugerens besked, find ud af, hvad brugeren har brug for
2. Efter at have identificeret intentionen, henvis sprogmodellen til et nyt system prompt, som enten kan hjælpe med problemet eller identificere en endnu mere specifik intention
3. Gentag trin 2 indtil sprogmodellen har sporet ind på, præcis hvad den skal gøre
4. Hjælp brugeren

Forneden er et simpelt eksempel:

System: Find ud af, om kunden skal have hjælp til:

- At finde informationer om hotellet
- Booke et ophold
- Lave en klage

User: Jeg har haft et forfærdeligt ophold!

System: Kunden vil gerne lave en klage. Du skal hjælpe ved <giv instruktioner>

Fordele ved at identificere brugerens hensigt:

- *Skræddersyet hjælp:* Metoden gør det muligt for sprogmodellen at give mere skræddersyet hjælp. Det kan føre til en bedre performance.
- *Mindre regnekraft:* Alternativt skulle en system message inkludere alle instruktioner for alle slags henvendelser, hvilket ville kræve større regnekraft og dermed være dyrere og muligvis langsommere.

⏳ Giv modellen tid til at "tænke"

Et stort problem, folk har med sprogmodeller, er at de let kan give forkerte informationer med stor selvsikkerhed. Vi kalder det “hallucinationer”, når sprogmodellen “finder på” et svar, der ikke er rigtigt. En simpel hjælp til at undgå hallucinationer er at bede sprogmodellen om ikke at give et forkert svar, hvis den ikke er sikker. Mange prompts, især ved egen programmering af chatbotter, inkluderer derfor en instruktion om, hvad den skal gøre, hvis den ikke kender svaret:

System: Hvis du ikke kender svaret, så sig: “Jeg kender ikke svaret”

Dette er dog ikke en fuldkommen løsning, og det kan i nogle tilfælde være nødvendigt at implementere andre logik, som kan hjælpe sprogmodellen med ikke at give forkerte svar, for eksempel ved at give den tid til at “tænke”. Det skal her siges, at sprogmodeller jo selvfølgelig ikke *kan* tænke. Sprogmodeller fungerer ved, at de forudsiger ord baseret på en masse tekstdata og dets mønstre, samt beregner ligheden mellem vektorer. Det er en matematisk og algoritmisk proces og derfor ikke en, som kræver en bevidsthed eller nogen form for forståelse for, hvad den egentlig producerer.

Mange af metoderne i denne sektion og de næste sektioner er et forsøg på at gøre sprogmodellen bedre til logik. Det ligger måske i navnet, “sprogmodel”, at det ikke er logik, som modellerne er stærkest indenfor. Sprogmodeller sætter ord og sætninger sammen, som den har set før. Det fungerer rigtigt godt, hvis man stiller den et spørgsmål, som allerede er i dens træningsdata, men det fungerer mindre godt, når den selv skal sammenstykke, hvad den ved om verdenen, for at finde dens eget svar. Et godt eksempel er, hvis man spørger den, hvad det tiende led i Fibonnaci-sekvensen er. Modellen ved helt sikkert hvordan man beregner sekvensen og har nok også set eksempler med endnu flere led end 10, men den har ikke nødvendigvis læst sætningen “det tiende led i Fibonnaci-sekvensen er 34”. Hvis den bruger logik, kan den let komme frem til svaret. I mange tilfælde kan prompt engineering hjælpe sprogmodellen ved at instruere modellen i at bruge logik eller ved at give den plads til det.

1. Bed modellen om selv at løse problemet, før den giver et svar

Sprogmodeller virker til at foretrække at være enig med brugeren. Hvis man kommer med et regnestykke og spørger, om det er korrekt, kan sprogmodellen derfor godt godkende regnestykket før, den egentlig har taget stilling til det. I stedet kan man bede sprogmodellen om at lave dens egne udregninger og dermed forhåbentligt give et mere korrekt svar.

System: Du vil blive præsenteret for en opgave og et svar. Du skal vurdere om svaret er korrekt eller forkert. Før du laver din vurdering, skal du komme med din egen udregning, da brugerens svar kan være forkert.

User: Opgaven: I 2024, hvad er aldersforskellen mellem den ældste og yngste præsident i USA, da de blev valgt?

Svar: Den ældste præsident er Joe Biden, som var 78, da han blev valgt.

Den yngste præsident var Theodore Roosevelt, som var 42.

80 - 42 = 38

Svaret er 38.

Assistant: Svar: Den ældste præsident er Joe Biden, som var 78, da han blev valgt.

Den yngste præsident var Theodore Roosevelt, som var 42.

78 - 42 = 36

Svaret er 36.

Svaret er forkert, da Joe Biden var 78 og ikke 80, da han blev valgt ind som præsident i USA. Det korrekte svar er 36 og ikke 38.

Nogle gange er det dog ikke nødvendigt, eller måske endda upassende, at dele sprogmodellens beregninger. For eksempel kunne man forestille sig, at eleven ikke ville have det rigtige svar til opgaven, men i stedet bare vide om svaret var korrekt eller forkert, så vedkommende kunne prøve igen. Vi kan tilføje til vores prompt, at svaret ikke skal afsløres:

System: Du vil blive præsenteret for en opgave og et svar. Du skal vurdere om svaret er korrekt eller forkert. Før du laver din vurdering, skal du komme med din egen udregning, da brugerens svar kan være forkert. Vis IKKE dine udregninger eller svaret.

Hvis brugerens svar er forkert, så giv et hint, der ikke afslører svaret.

2. Spørg modellen, om den er sikker

Vi kan bede sprogmodellen om at genoverveje dens svar i en række kontekster. Hvis vi for eksempel ikke er sikre på, om svaret er rigtigt, kan vi spørge modellen, om den er sikker på svaret. Hvis vi har bedt den om at bruge konteksten fra en mængde dokumenter, kan vi spørge den, om den har analyseret alle dokumenterne. Hvis vi har bedt den om at lave en opsummering, kan vi spørge, om den har lavet den bedste opsummering, som den kan.

Et problem med denne tilgang er dog, at modellen nogle gange ændrer sit svar fra et godt eller korrekt svar til et dårligere et, fordi den vurderer, at brugeren ikke er tilfreds med svaret. Derfor skal man passe på, når man bruger denne metode. Måske har du oplevet noget lignende, hvis nogen har spurgt dig, om du er sikker på dit svar?

3. Maieutic/Socratic prompting

Maieutic eller Socraitic prompting er en metode, hvorved man vedholder en længere dialog med sprogmodellen. Metoden bygger på den Sokratiske undervisningsmetode, hvor man i stedet for at give svaret til en elev stiller spørgsmål og dermed guider eleven til at tænke dybere over spørgsmålet og selv komme frem til et svar.

I forhold til sprogmodeller fungerer metoden ved, at når modellen giver et svar, så spørger man ind til svaret. Man kan for eksempel bede den om at fortælle mere eller at uddybe dele af dens svar. Dermed kan modellen komme frem til et bedre og mere uddybende svar, og den kan i nogle tilfælde indse, at dens originale svar var ufuldkomment eller endda forkert.

4. Self-Refinement prompting

I self-refinement prompting forbedrer sprogmodellen selv sit output ved at give feedback på dens svar og tilpasse svaret baseret på dens egen feedback. Det er en simpel metode, som man kan gøre i et loop, enten en bestemt mængde gange, eller til et mål er opnået. Målet kunne for eksempel være, at sprogmodellen ikke har mere at tilføje.

System: Du skal skrive et kort svar på et spørgsmål, som ikke er mere end tre sætninger. Efter du har svaret, skal du give feedback på dit svar og omskrive dit svar baseret på feedbacken. Du skal give feedback og rette dit svar til, indtil du ikke har mere feedback at give.

User: Hvorfor er motion vigtigt for helbredet?

Assistant: Svar: Motion er vigtigt for helbredet, fordi det forbedrer den fysiske formåen,øger energi og forebygger kroniske sygdomme. Det hjælper også med at reducere stress og forbedre mental sundhed. Regelmæssig motion kan desuden fremme en sund vægt
og forbedre livskvaliteten.

Feedback: Svaret giver en god oversigt, men det kan være mere præcist ved at nævne specifikke sygdomme, som motion kan forebygge. Det kunne også være godt at inkludere, hvordan motion påvirker hjertet eller kredsløbet.

Omskrevet svar: Motion er vigtigt for helbredet, fordi det forbedrer den fysiske formåen og forebygger kroniske sygdomme som type 2-diabetes, hjertekarsygdomme og fedme. Det øger også energiniveauet og reducerer stress, som er gavnlige for mental sundhed. Regelmæssig motion fremmer desuden en sund vægt
og forbedrer livskvaliteten.

Feedback: Det nye svar er mere informativt, men det kunne være endnu mere præcist ved at nævne specifikke fordele ved motion, som styrkelse af muskler og knogler samt
forbedring af kredsløbssystemet.

Omskrevet svar: Motion er vigtigt for helbredet, fordi det forbedrer den fysiske formåen, styrker muskler og knogler samt forebygger kroniske sygdomme som type 2-diabetes og hjertekarsygdomme. Det øger energiniveauet og reducerer stress, hvilket er gavnligt for mental sundhed. Regelmæssig motion fremmer desuden en sund vægt og forbedrer livskvaliteten.

Feedback: Svaret er nu præcist og informativt. Der er ikke mere feedback, da det dækker emnet godt.

Madaan et al. 2023 har testet metoden på outputtet af syv opgaver, for eksempel at optimere kode og omskrive anmeldelser til det modsatte synspunkt. Alle outputs blev forbedret med mellem 5%-40%.

📖 Brug af relevant vidensgenerering

En mængde teknikker bruger vidensgenerering til at forbedre sprogmodellens svar. Det kan se lidt forskelligt ud, som vi vil få at se i de næste metoder, men fælles for dem alle er, at sprogmodellen genererer relevante tekster og informationer, før den løser dens opgave.

At generere relevant viden fungerer meget i samme stil som metoderne, hvor sprogmodellen får tid til at "tænke". Det vil sige, at modellen ved først at generere viden, sporer sig selv bedre ind på, hvad den skal bruge til at svare på spørgsmålet, og derfor har en bedre chance for at svare rigtigt og godt. Det kan svare lidt til, hvis man før man begynder at skrive en stil, sætter sig ned og brainstormer omkring emnet og skriver informationer ned, som man skal huske at inkludere.

1. Generated Knowledge Prompting

I Generated Knowledge Prompting beder vi først sprogmodellen om at generere relevante informationer, før vi beder den om at udføre en opgave. Det kan for eksempel være som forneden:

System: Skriv fem grunde til, at det kan være nyttigt at lære om prompt engineering.

Bagefter, skriv et spændende, men kort LinkedIn post om fordelene ved at lære prompt engineering, hvor du inkluderer de tidligere grunde.

Denne metode kan enten udføres i et prompt som foroven, eller i flere prompts, hvor man først spørger sprogmodellen om at generere fakta og bagefter spørger den om at skrive en tekst ud fra det. Man kan også bruge metoden til at besvare fakta-baserede spørgsmål. For eksempel kan den svare på, hvor hurtig en løve og tiger kan løbe, før den svarer på, hvilken er hurtigst.

2. Step-Back Prompting

I Step-Back prompting beder vi sprogmodellen om at tage et "skridt tilbage" og lave nogle refleksioner om viden, der kunne være relevant for spørgsmålet. Det kunne man for eksempel gøre som forneden:

System: Du vil lige om lidt blive spurgt om et spørgsmål. Når du svarer på spørgsmålet, skal du gøre det i 3 trin:

1. Bestem hvilke koncepter og viden er relevant for at besvare spørgsmålet
2. Brug koncepterne og viden til at besvare spørgsmålet
3. Besvar spørgsmålet

3. Recitation-Augmented Language Models

Recitation-Augmented Language models er en teknik, hvor modellen henter, eller reciterer, relevante informationer som del af dens svar. Den her metode er forskellig fra de to foroven, fordi den kræver, at man har givet sin sprogmodel en stor mængde træningsdata. I Sun et al. 2023, hvor metoden først blev introduceret, havde de for eksempel trænet deres model på data fra Wikipedia. Standard online sprogmodeller såsom ChatGPT kan ikke recitere passager fra deres træningsdata.

Metoden involverer to trin:

1. Sprogmodellen finder dokumenter i dens træningsdata, som kan hjælpe den med at besvare spørgsmålet. Hvis der er mange, kan en algoritme hjælpe med at vælge de bedste passager
2. Sprogmodellen citerer en eller flere relevante passager og kommer derefter med dens svar

Fordel

- Recitation-Augmented Language models er især nyttige i situationer, hvor det er vigtigt at kunne dobbeltchecke, at svaret er rigtigt. Hvis brugeren kan se, hvad modellen har baseret svaret på, er det lettere at verificere, om det er et godt svar.

Problem

- Det er svært og tager lang tid at opdatere træningssættet, da modellen så skal trænes forfra. Hvis træningssættet ikke bliver opdateret, kan informationen indenfor nogle områder være forældet.

Chain of Thought

Chain of Thought er endnu en metode, hvor man hjælper sprogmodellen til at "tænke". Her sker det ved at man beder modellen om at "tænke højt", altså at vise og forklare dens tankegang. Det er en af de mest populære metoder, hvis ikke den *mest* populære, til at forbedre sprogmodellers svar.

Hvorfor er det nyttigt?

1. *Tankeprocess:* Det hjælper dig med at forstå, hvordan sprogmodellen når frem til sine konklusioner. Det er en spændende mulighed for at lære om dens tankeprocess!
2. *Fejlfinding:* Det kan afsløre eventuelle fejl i sprogmodellens ræsonnement. Det gør det også lettere for dig at identificere hvor, den er gået galt, og pege den i rette retning
3. *Bedre svar:* Det giver ofte bedre, mere detaljerede og gennemtænkte svar

Simpel implementering

At implementere Chain of Thought er i teorien så simpelt som at skrive en enkelt instruks, hvor man beder modellen om at dele dens "tankeprocess". Det kunne man for eksempel gøre som forneden:

System:	Forklar, trin for trin, hvordan man finder kvadratroden af 256

Avanceret eksempel:

Der findes mange og mere advancerede implementeringer af Chain of Thought end den foroven. Her viser vi en implementering, som kræver flere trin:

1. Start med at definere problemet klart.
2. Bed AI'en om at opdele problemet i mindre dele eller trin.
3. Anmod om en forklaring for hvert trin.
4. Bed om en endelig konklusion baseret på de gennemgående trin.

Eksempel:

System: Lad os løse følgende problem trin for trin: En butik sælger T-shirts for 150 kr. stykket. Hvis man køber 3 eller flere, får man 20% rabat på hele købet. Alice køber 4 T-shirts. Hvor meget skal hun betale?

1. Identificer først de relevante oplysninger i problemet.
2. Beregn den oprindelige pris for 4 T-shirts uden rabat.
3. Afgør om Alice er berettiget til rabatten og forklar hvorfor.
4. Beregn rabatbeløbet.
5. Træk rabatten fra den oprindelige pris for at finde den endelige pris.
6. Giv det endelige svar og opsummer beregningen.

Venligst vis din tankegang for hvert trin.

Forneden introducerer vi overbygninger til Chain of Thought, som kan gøre metoden endnu bedre.

1. Self-consistency metode med Chain of Thought

I denne metode beder man også sprogmodellen om at vise sin tankegang, men man genererer flere svar. Derefter får man modellen til at vælge det bedste svar, fra de svar, den har genereret. Det gør man ofte ved "majority rule", hvor den vælger det svar, der er blevet genereret flest gange, men man kan også specificere en anden standard. For eksempel, hvis svaret er en tekst generering, kan man få den til at vurdere det bedste svar ved at sammenligne hvilke temaer og formuleringer, der er gennemgående.

System: Du skal finde ud af, hvad 12 ganget med 3 er. Du skal vise din tankegang, mens du laver udregningerne. Gør dette 4 gange og vælg derefter det svar, som du får mest ofte.

Assistant: Her er mine udregninger:

1. 10x3 = 30; 2x3 = 6; 30 + 6 = 36
2. (10 + 2)x3 = (10x3) + (2x3) = 30 + 6 = 36
3. 10x3 = 30; 30 x2 = 60
4. 12 er 3x4 så 12x3 er 3x(4x3) = 3x12 = 36

1,2, og 4 giver alle 36, så 36 er svaret.

Hvis problemerne er meget komplekse og sprogmodellen har brug for meget plads til at vise sin tankegang, kan man i stedet splitte sin prompt op og spørge sprogmodellen om at komme med svar af flere omgange, før man beder den om at sammenligne svarene og finde det bedste.

2. Complexity-based prompting

Denne metode går ud fra, at når vi genererer et svar ved hjælp af Chain of Thought, så performer svar med flere "reasoning steps", det vil sige flere mellemtrin, generelt bedre. Den teori er blevet testet i for eksempel Fu et al. (2023), hvor svar med flere mellemtrin gennemsnitlig var mere nøjagtige. Man kan implementere dette på to måder:

- Man kan bruge complexity som et udvælgelseskriterium i self-consistency metoden
- Man kan bede sprogmodellen om at generere mange eller en specificeret mængde mellemtrin

3. Chain of Thought med og uden eksempler

Eksemplerne vi hidtil har beskrevet er **zero shot** eksempler på Chain of Thought, hvilket betyder at vi ikke har givet sprogmodellen nogle eksempler på hvilke mellemtrin, vi forventer. Når man kombinerer Chain of Thought med **few shot** prompting, det vil sige, man giver den eksempler på tankegangen og mellemtrinene, man gerne vil have den til at generere, så hedder det **Manual Chain of Thought**. Manual Chain of Thought performer tit bedre end Zero Shot Chain of Thought, men kræver til gengæld, at man laver eller har eksempler, man kan give sprogmodellen.

Et alternativ til Zero Shot og Manual Chain of Thought er **Auto Chain of Thought**. I Auto Chain of Thought genererer sprogmodellen selv sine egne eksempler, som den kan bruge til at træne sig selv. I stedet for både at præsentere sprogmodellen for et spørgsmål og et svar eksempel, præsenterer man den kun for en mængde spørgsmål. Ud fra de spørgsmål genererer den selv svar med mellemtrin som en iterativ Zero Shot Chain of Thought. Svarene med mellemtrin, den har genereret, bruger den til sidst som eksempel til at svare bedst muligt på det spørgsmål, man egentlig ønsker svar på. Man kan tænke det som om, at sprogmodellen øver sig, før den rigtigt svarer.

Trin i Auto Chain of Thought

- Man giver sprogmodellen en mængde spørgsmål og beder den svare på spørgsmålene ved at vise sin tankegang
- Spørgsmålene, tankegangen og svarene bliver fordret tilbage til sprogmodellen som eksempler
- Sprogmodellen svarer på et nyt spørgsmål og viser sin tankegang

Auto Chain of Thought minder meget om en anden metode, som hedder **Analogical prompting**. I Analogical prompting beder man også modellen om at generere eksempler før, den svarer, men her finder modellen også på sine egne spørgsmål. Man skal derfor hverken have spørgsmål eller svar forberedt i den metode. Research peger på, at det er ideelt at bede modellen om at generere 3-5 svar-spørgsmål par, før den svarer.

Least-to-Most Prompting

Least-to-Most prompting er et alternativ til Chain of Thought, som kræver en smule mere hjælp fra brugeren, men som til gengæld kan være bedre til komplekse problemstillinger. Denne metode er især god til tekstbaserede matematik problemer.

I Least-to-Most prompting spørger man først sprogmodellen, hvilke underproblemer, den skal løse, for at løse det overordnede problem. Efter sprogmodellen har identificeret underproblemerne, løser man dem en efter en. Typisk sker dette ved hjælp af en række prompts, hvor svarene til hver underproblem bliver en del af det nye prompt, men man kan også gøre det i et samlet prompt. Forneden er et eksempel:

System: Du får matematik problemer, som du skal løse. Du skal løse dem ved først at lave en liste over underproblemer, som skal løses. Derefter skal du løse hvert problem og til sidst komme med svaret til opgaven. Gå igennem problemet trin for trin.

User: En landmand har en mark, der er 120 meter lang og 80 meter bred. Han skal plante træer langs markens omkreds. Hvert træ plantes 5 meter fra hinanden. Hvor mange træer skal han plante?

Assistant: Underproblemer:

1. Beregn markens omkreds
2. Find ud af, hvor mange træer, der kan plantes langs omkredsen, når de skal plantes 5 meter fra hinanden

Løsning af underproblemer:

1. Beregn markens omkreds:

Markens omkreds O kan beregnes ved hjælp af formlen:

$O=2\times(længde+bredde)$

Hvor længden er 120 meter og bredden er 80 meter.

$O=2\times(120m+80m)=2\times200m=400m$

2. Find ud af, hvor mange træer der kan plantes langs omkredsen:

For at finde antallet af træer N, som skal plantes, bruger vi følgende formel:

$N= O/(afstand\ mellem\ træer)$

Afstanden mellem træer er 5 meter, så vi kan beregne:

N= 400 m/ 5m = 80

Samlet løsning: Landmanden skal plante 80 træer langs markens omkreds.

Ekspert

I den her sektion præsenterer vi metoder, hvor vi inkluderer eksterne redskaber som for eksempel en anden (sprog)model til at forbedre vores prompts. Flere af metoderne kræver kompetencer indenfor kodning. Det er ikke en fuldkommen gennemgang af metoderne, da man kunne skrive en længere guide til hver, men metoderne her beskrevet kan forhåbentlig give inspiration til videre research.

📂 Load data ved hjælp af embeddings

Vi har allerede nævnt hvordan sprogmodellers begrænsede "context window" betyder at store mængde tekster ikke kan blive tilføjet som input af en bruger eller i en prompt. Det kan for eksempel være nyttigt at uploade en kilde med opdateret og troværdigt information, eller at gemme lange tidligere samtaler i prompten som kontekst for nye samtaler. Se evt. Del komplekse opgaver op i mindre dele og Giv reference tekster.

For at komme omkring denne begrænsning, kan man i stedet konvertere sin tekst til "embeddings". Embeddings er en konvertering af tekst til vektorer, som sprogmodeller kan bruge til at sammenligne afstande for at identificere semantiske ligheder mellem ord, sætninger eller tekster. Der er en række modeller, man kan bruge til at konvertere tekst til vektorer, som kan gemmes i vector databaser som for eksempel chromadb. Her er en liste, der kan give et overblik over gode databaser.

Denne metode bliver for eksempel brugt til Retrieval-Augmented Generation (RAG) chatbots, som er meget populære. En RAG chatbot kombinerer styrken fra offentlige sprogmodeller, oftest OpenAI's, med specialiseret viden fra dokumenter loadet med embeddings. Man kunne for eksempel programmere en RAG, som er ekspert indenfor jura eller Danmarks elektriker og vvs-branche.

Nogle fordele ved RAGs

- *Kontekstuelle svar:* RAG-chatbots kan give mere relevante svar, fordi de er trænet indenfor et specifikt område.
- *Opdateret information:* RAG-chatbots kan let blive opdateret, så de har det nyeste data og informationer. Offentlige sprogmodeller bliver ikke lige så tit opdateret.
- *Korrekt information:* RAG-chatbots kan give mere korrekte oplysninger, da de ofte er trænet på data, som er blevet specielt udvalgt, og derfor kan være mere troværdig.
- *Links*: RAG-chatbot kan ofte give links til data, de har brugt til at besvare et spørgsmål. Det gør det lettere at dobbelt-checke vigtig information og bruge svaret til videre research.
- *Personaliserede:* RAG-chatbots er specialiserede indenfor specifikke områder. Man kan derudover give chatbotten specifikke informationer eller værktøjer, som kan gøre den bedre til at løse en given opgave.
- *Fejlhåndtering:* RAG-chatbotter er bedre til at indrømme, hvis de ikke ved noget, og de ikke kan give et godt svar. Her finder generelle sprogmodeller ofte på et svar, når de ikke har nok information.

Det er ikke så besværligt at programmere sin egen RAG, og man kan finde flere gode tutorials online, for eksempel den her.

💻 Implementer kode i din prompt

Sprogmodeller får ofte kritik for at fejle selv simple opgaver. For eksempel har mange forgæves forsøgt at få ChatGPT til korrekt at tælle mængden af bogstavet "r" i ordet "strawberry". Det er fordi, at sprogmodeller fungerer ved, at de gætter det næste token, altså det næste bogstav eller ord. Det er en *sprog*model, og kan derfor ikke lave udregninger selv. Den kan derfor for eksempel komme til at gætte, at der er 2 r'er i stedet for 3. Dog ville en simpel python kode let kunne udregne hvor mange r'er, der er i ordet "strawberry":

```
r_tæller = "strawberry".count("r")
```

Sprogmodeller kan ikke eksekvere kode. Dog kan man, hvis man programmerer sin egen sprogmodel, give den ekstra funktionalitet i form af kode. Koden bliver ikke eksekveret af sprogmodellen, men for eksempel Python interpreter kan køre koden, hvis man har givet modellen adgang til det.

System: Du kan bruge Python kode til at svare på et spørgsmål. Python koden er omringet af tre backticks. Brugerne vil gerne vide, hvor mange bogstaver der er i et ord. Her er en kode, du kan bruge:

letter_counter = word.count(letter)

Lad ordet være "word" og bogstavet være "letter". Giv svaret i "letter_counter".

I prompten kan du også kalde eksterne API'er og bruge det i en kode eksekvering. Der skal du dog passe på, at du ikke kører noget kode, som kan skabe problemer for dig eller din bruger.

🌐 Lav kald til eksterne systemer

Der findes flere indbyggede funktioner, som man kan kalde i sit prompt. Et eksempel er i OpenAI, hvor man kalder eksterne systemer med "function calling".

OpenAI identificerer fem kontekster, hvor det kan være nyttigt at bruge deres function calling:

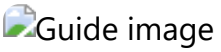
1. *Få assistenter til at hente data:* Hvis en bruger spørger en kundeservice chatbot om for eksempel, hvad deres sidste ordrer var, så kan en AI assistent hente data fra et internt system og besvare spørgsmålet
2. *Lad assistenten tage handlinger:* Hvis en AI kan tage handlinger, så kan den for eksempel booke et møde for brugeren eller bestille en vare.
3. *Lad assistenter lave udregninger:* En sprogmodel er ikke særlig god til at lave udregninger. I stedet kan den forbinde til en matematik funktion, der kan hjælpe.
4. *Byg gode arbejdsprocesser:* AI'en kan bruge en dataudtrækningspipeline, der indsamler rå tekst, konverterer den til strukturerede data og gemmer den i en database.
5. *Modificer din UI:* Et function call kan opdatere din apps UI baseret på, hvad brugeren skriver. Det kunne for eksempel være at sætte en ny lokation på et kort.

Ligesom i den sidste metode, hvor vi implementerer kode, så kan sprogmodellen ikke selv implementere noget af det foroven. Det kræver at eksterne systemer kan implementere det for modellen. Det er kun modellens job at identificere og bestemme hvilke af funktionerne, der skal implementeres.

Tree of Thought

Tree of Thoughts er en prompting metode, som bygger ovenpå Chain of Thought.

I stedet for at bede sprogmodellen om at evaluere tankegangen i et eller flere svar, evaluerer vi her hvert trin i problemløsningen. Hvis modellen har gået galt i et trin, så kan vi gå tilbage til et tidligere trin. Det fører til en matrix baseret tilgang i sammenligning med den lineære tilgang i Chain of Thought. Nedenfor er en illustration af Yao et al. (2023), som først foreslog metoden:



I illustrationen ser vi:

- Normal input-output prompting, det vil sige standard prompting uden Chain of Thought eller Tree of Thought, genererer et output uden nogle mellemtrin.
- I Chain of Thought tilføjer vi mellemtrin i form af en synlig tankegang.
- Når vi tilføjer Self Consistency, begynder sprogmodellen også at overveje alternative måder at løse problemet.
- Tree of Thoughts udforsker flere muligheder for hver mellemtrin, eller "tanke", og vurderer hvor "god" hver tanke er, før den fortsætter. I diagrammet er "gode" tanker i grønne nuancer og "dårlige" tanker i røde nuancer. Hvad der konstituere en "god" eller "dårlig" tanke skal specificeres for modellen, så den ved, hvordan den skal bedømme dens output.

Tree of Thoughts er inkluderet i ekspert sektionen fordi den normalt bliver implementeret ved hjælp af kode. Koden sender flere prompts til sprogmodellen og beder den stoppe efter hver trin for at evaluere dens output. Hvis man ikke selv vil programmere en iterativ Tree of Thought prompter, kan man bruge et eksisterende bibliotek, for eksempel her eller her.

En mere simpel version af Tree of Thought kræver dog ikke programmeringsevner. Dave Hulbert har implementeret Tree of Thought logik ved at give sprogmodellen et eksempel med tre eksperter. Forneden har vi oversat hans originale prompt til dansk:

System: Der er tre eksperter, som skal besvare et spørgsmål.

Alle eksperter skriver ned, hvad de tænker i forhold til det første skridt og deler det med gruppen.

Derefter fortsætter eksperterne til de næste skridt, indtil de finder et svar.

Hvis en af eksperterne indser, at de har taget fejl, går vedkommende.

Spørgsmålet er ...

Tree of Thought kan være særlig god til komplekse opgaver, men er unødvendig for simple opgaver.

🧠 Meta Prompting til at lave nye prompts

Det kan være tidskrævende at skrive perfekte prompts. I stedet kan du få din sprogmodel til at lave sine egne prompts. En meget let implementering ville være at beskrive en opgave til sprogmodellen og bede den om at skrive en bedre prompt. Det er dog ikke sikkert, at sprogmodellen er særlig god til at skrive prompts, og derfor kræver den nok en del instruktion.

Der er online prompts generators, som er trænet til at omskrive prompts, så man kan få mest muligt ud af sprogmodellen. Der er gratise prompt generators, for eksempel den her, og også nogle, der koster lidt, som den her.

Selvfølgelig ville det være meget sjovere at instruere vores egen prompt generator, hvilket vi kan gøre igennem meta prompting. Da dette kræver mange instruktioner, ofte i et loop, bliver den her metode oftest implementeret igennem programmering. Et eksempel på sådan kode kan findes her. Et andet eksempel er Anthropics kode, som også er hurtig og let at implementere selv.

✅ Valg af gode eksempler og Active Prompting

Hvis man inkluderer eksempler i for eksempel sin Chain of Thought prompt (se: Chain of Thought med og uden eksempler), er det ikke ligemeget hvilke eksempler, det er. Det er der flere grunde til:

1. *Upassende eksempler*

Det kan være, at du vil have modellen til at løse et komplekst problem, men har givet simple eksempler, eller modsat. Det kan også være, den har fået eksempler fra tekst-genererings opgaver, men skal løse en matematisk opgave. I sådanne tilfælde kan eksemplerne forvirre modellen og endda forværre outputtet, fordi den tror, at den skal bruge en upassende fremgangsmåde til at løse problemet

2. *Forkerte eksempler*

På samme måde er det ikke ideelt, hvis modellen får eksempler, hvor der er fejl. Det kan være, at tankegangen eller svaret er forkert, og det kan også lede modellen på afveje. Man kan sørge for, at eksemplerne er korrekte ved at have mennesker til at generere eller annotere dem, som i Manual Chain of Thought. Det kan dog være meget tidskrævende og dyrt at lave eller checke mange eksempler.

Active Prompting

Active Prompting er blevet foreslået som en løsning til iterativt at mindske forkerte eksempler. Det gør den ved følgende metode:

1. Uncertainty Estimation:

- Sprogmodellen bliver præsenteret for en mængde spørgsmål, som den skal træne på.
- Den besvarer alle spørgsmålene med Chain of Thought en hvis mængde gange.
- Spørgsmålene bliver scoret på, hvor usikker modellen er på svaret. Det kan for eksempel være, at svar hvor modellen har fået det samme svar 4 ud af 5 gange er 20% usikker. Andre metoder kan også bruges, dog skal man huske at det ikke er særlig værdifuldt at spørge modellen om, hvor sikker den er, da sprogmodeller generelt fremstår meget sikre.

2. Selection:

- Spørgsmålene, hvor modellen er mest usikker, bliver udvalgt

3. Annotation:

- De udvalgte spørgsmål bliver annoteret, det vil sige rettet, manuelt

4. Final Inference:

- De rettede usikre svar samt de mere sikre svar kan bruges som trænings eksempler i en Chain of Thought prompt

På den måde kan modellen generere langt de fleste af dens egne eksempler og kun de eksempler, hvor den virker usikker og laver fejl, bliver rettet til manuelt af et eller flere mennesker. Det sparer tid og dermed også penge. Til gengæld koster det regnekraft og mange tokens og dermed også penge at finde frem til spørgsmålene, der skal annoteres.

🧪 Lav systematiske ændringer og test det

Hvordan ved man, om ens prompt ændringer har gjort modellens svar bedre eller værre? Hvad hvis man har tilføjet en prompt, som forhøjer den nødvendige regnekraft, eller får modellen til at performe langsommere, men dette fører ikke til en stor forbedring i svar? Man kan selvfølgelig kigge på svarene individuelt og vurdere deres kvalitet, men hvis man har en model, som skal tilgås af mange brugere med en bred vifte af spørgsmål, så kan det give mere mening at lave systematiske tests.

Hvordan tester jeg bedst min sprogmodel?

- *Videnskabelig tilgang:* Det er vigtigt, når man tester, at man kun laver få ændringer af gangen, så man kan gå tilbage og finde ud af, hvilke prompts der har været nyttige og ikke nyttige.
- *Kvalitetsvurdering:* Man kan teste ens model ved at spørge den en række spørgsmål med klare svar eller ved at bruge en anden model til at vurdere kvaliteten.
- *Svarudvalg:* Spørgsmålene skal være brede og repræsentative for de spørgsmål, brugerne kan finde på at stille, og der skal være mange af dem, så resultaterne bliver statistisk signifikante.

Nedenunder er *et* eksempel på, hvordan man kan bruge en prompt til at evaluere et svar:

System: Du vil blive præsenteret for et spørgsmål, et ideelt svar, og et svar, som du skal vurdere ("vurderings svar").

Trin 1: Skriv tre punkter for, hvad du vurderer at være de tre vigtigste informationer i det ideelle svar baseret på spørgsmålet.

Trin 2: For hver punkt, vurder hvorvidt vurderings svaret kommer ind på punktet. Skriv "ja" udfra punktet, hvis punktet bliver dækket korrekt.

Trin 3: Tæl hvor mange "ja'er", der er, og giv det som en endelig score. Skriv: {score: <score>}

User: Spørgsmål: Hvem er i den danske kongefamilie, og hvem er regenten i 2024?

Ideelt svar: Det nuværende danske kongehus ledes af kong Frederik 10. og dronning Mary, som er hans gemalinde. De har fire børn: kronprins Christian, prinsesse Isabella samt tvillingerne prins Vincent og prinsesse Josephine. Dronning Margrethe II abdicerede den 14. januar 2024, men forbliver en del af kongehuset. Desuden inkluderer familien kongens

yngre bror, prins Joachim, og hans fire børn: grev Nikolai, grev Felix, grev Henrik og komtesse Athena, samt dronning Margrethe II.s søster, prinsesse Benedikte.

Vurderings svar: Den danske kongefamilie består af dronning Margrethe II, kronprins Frederik, kronprinsesse Mary, deres børn prins Christian, prinsesse Isabella, prins Vincent og prinsesse Josephine, samt andre medlemmer af familien. I 2024 er dronning Margrethe II regent. Hun har været dronning siden 1972 og er den første kvindelige monark i Danmark i over 500 år.

Assistant: Her er min vurderinger:

1. Kong Frederik 10. og dronning Mary leder kongehuset og har fire børn: kronprins Christian, prinsesse Isabella samt tvillingerne prins Vincent og prinsesse Josephine.

Vurdering: Vurderings svaret nævner ikke kong Frederik 10. som regent eller forbindelsen til dronning Mary.

2. Dronning Margrethe II abdicerede den 14. januar 2024, men forbliver en del af kongehuset.

Vurdering: Vurderings svaret nævner ikke abdikationen af dronning Margrethe II.

3. Kongehuset inkluderer også prins Joachim og hans fire børn samt prinsesse Benedikte.

Vurdering: Vurderings svaret nævner ikke prins Joachim eller prinsesse Benedikte.

{score: 0}