Høgskulen på Vestlandet

BACHELOROPPGAVE

Anbudsassistent - Generering av anbud med kunstig intelligens

Tender assistant - Tender generation using artificial intelligence

Martin Berg Alstad

Nora Kristiansen

Torbjørn Vatnelid

Informasjonsteknologi

Institutt for datateknologi, elektroteknologi og realfag

Fakultet for teknologi, miljø- og samfunnsvitenskap

Carsten Gunnar Helgesen

Innleveringsdato 13.05.2024

Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap

Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel:	*Dato:*
Anbudsassistent – generering av anbud med kunstig intelligens	*13.05.2024*
Forfatter(e):	*Antall sider u/vedlegg: 80*
Martin Berg Alstad, Nora Kristiansen, Torbjørn Vatnelid	*Antall sider vedlegg: 1*
Studieretning:	*Antall disketter/CD-er:*
Informasjonsteknologi	0
Kontaktperson ved studieretning:	*Gradering:*
Carsten Gunnar Helgesen	Ingen
Merknader:	

Oppdragsgiver:	*Oppdragsgivers referanse:*
Link Utvikling AS	
Oppdragsgivers kontaktperson:	*Telefon:*
Michelle Sæther	902 73 990

Sammendrag:

Dette bachelorprosjektet omhandler utvikling av applikasjonen Anbudsassistent, utviklet på oppdrag fra Link Utvikling. Prosjektet har fokusert på å utvikle en applikasjon som automatisk genererer anbud ved hjelp av kommersielt tilgjengelige store språkmodeller.

Stikkord:

Maskinlæring og store språkmodeller	Prompt Engineering	Anbudskonkurranser

Høgskulen på Vestlandet, Fakultet for teknologi, miljø- og samfunnsvitenskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

FORORD

Rapporten gjennomgår prosjektet Anbudsassistent - Generering av anbud med kunstig intelligens. Bachelorprosjektet markerer slutten på vår treårige bachelor i Informasjonsteknologi ved Høgskolen på Vestlandet.

Vi ønsker å takke alle som har hjulpet oss underveis i prosjektets løp og gjort at prosjektet ble vellykket. Først og fremst vil vi rette en stor takk til vår veileder hos Høgskolen på Vestlandet, Carsten Gunnar Helgesen, for god veiledning og tilbakemelding rundt prosjektgjennomføring og utarbeiding av denne rapporten. Takk til alle hos Link Utvikling, Michelle Sæther, Hana Colakovic, Pål Vårdal Gjerde og ikke minst hunden Leon for uvurderlig hjelp, veiledning og støtte underveis. Ikke minst vil vi takke eksterne brukertestere fra FunBit, Marius Nesse, og fra Rainfall, Børre Jensen og Nils Magne Lunde, som generøst brukte tiden sin til å gi gode tilbakemeldinger om applikasjonen.

Til sist ønsker vi å takke hverandre for et gjensidig godt samarbeid gjennom prosjektarbeidet.

Ordliste

Ordliste anbud

Ord	Betydning
Anbudskonkurranse	En konkurranse utlyst av en bedrift for å finne riktig kompetanse til en jobb.
Anbud	Et bindende tilbud som svar på en anbudskonkurranse.
Statens standardavtaler (SSA)	Statens kontraktsmaler for offentlige anbudskonkurranser.
Doffin	Nasjonal kunngjøringsdatabase for offentlige anskaffelser
Bistandsavtalene (SSA-B og SSA-B enkel)	Avtaler for kjøp av konsulentbistand (Doffin, Database for offentlige anskaffelser, 2024a).
Utviklings- og tilpasningsavtalen (SSA-T)	Avtale for kjøp av programvare som skal utvikles eller tilpasses kunden (Doffin, Database for offentlige anskaffelser, 2024b).
Kompetansematrise	Tabell som beskriver kompetansen til en eller flere ansatte.

Ordliste maskinlæring

Ord	Betydning
Kunstig intelligens (KI)	Informasjonsteknologi som justerer sin egen utdata og derfor fremstår som intelligent (Tidemann, 2024). En motsetning til regeldrevne applikasjoner.
Nevrale nettverk	En undergren av maskinlæring, inspirert av menneskehjernen.
Stor språkmodell (Large Language Model)	En avansert maskinlæringsmodell som spesialiserer seg på naturlig språk.
Vekt	Et trenbart parameter i nevrale nettverk, en verdi som blir multiplisert med utdata fra en node.
Bias	Et trenbart parameter i nevrale nettverk, en verdi som blir lagt til summen av all inndata til en node.
Aktiveringsfunksjon (Activation Function)	En funksjon som bestemmer om en node skal bli aktivert (sende verdien sin videre) eller ikke.
ReLU	En aktiveringsfunksjon som setter all negativ inndata til 0, og beholder positiv inndata slik den er.
Fremoverpropagering (Forward Propagation)	Prosessen med å generere en prediksjon ved å la data gå fra inndata-laget til utdata-laget.
Etikett (Label)	Det korrekte svaret på et datapunkt i et treningssett.
Tapsfunksjon (Loss Function)	Måler feilmarginen mellom prediksjon og etikett.

Ord	Betydning
Kryssentropi	Måler differansen mellom to sannsynlighetsfordelinger: den predikerte og den reelle.
Gradientnedstigning (Gradient Descent)	Optimaliseringsalgoritme for maskinlæringsmodeller som forsøker å minimere tapsfunksjonen.
Bakoverpropagering (Backpropagation)	Prosessen med å oppdatere vekter og biaser i et nevralt nettverk ved hjelp av gradientnedstigning.
Epoke (Epoch)	En fullføring av fremoverpropagering og bakoverpropagering under trening av nevrale nettverk.
Eksploderende gradienter (Exploding gradients)	Når vektene som nettverket oppdateres med blir eksponensielt større jo lengre bak i nettverket man kommer.
Forsvinnende gradienter (Vanishing gradients)	Når vektene som nettverket oppdateres med blir mindre og mindre jo lengre bak i nettverket man kommer.
Token	En enhet av tekst som kan sendes til eller produseres av en stor språkmodell.
Subword tokenization	En måte for tokenisering der hver token kan være et delord. Gjør det lettere å sette sammen kompliserte og sjeldne ord.
Embedding	En prosess for å omgjøre tekst til en vektor av tall som representerer teksten.
Kontekstvindu (Context window)	Grensen for hvor mange tokener en stor språkmodell kan bruke totalt på inndata (prompt + metadata) og utdata fra modellen sin respons.
Posisjonskoding (Positional encoding)	Metode for å unikt representere hver posisjon i modellens kontekstvindu.
Selv-oppmerksomhet (Self-attention)	En mekanisme brukt i store språkmodeller som gjør at modellen kan finne relasjoner mellom ord.
Flerhodet oppmerksomhet (Multi-head attention)	Flere mindre selv-oppmerksomhets-hoder som fokuserer på ulike kvaliteter ved tekst.
Transformator-arkitekturen (Transformer architecture)	Arkitektur for store språkmodeller som benytter selv-oppmerksomhet for å forstå og generere tekst.
Fremovermating-lag (Feed forward layer)	Fullt tilkoblede nevrale nettverk der all dataflyten skjer i én retning.
Residual-kobling (Residual connection)	Lar data hoppe over lag i nettverket.
Temperatur (Temperature)	En parameter som justerer tilfeldighet i valg av neste token i en sekvens. Høyere temperatur fører til mer tilfeldighet.

Ord	Betydning
Forhåndstrent (Pre-trained)	En forhåndstrent modell er gjerne trent på store mengder data og kan utføre et vidt spekter av oppgaver.
Finjustering (Fine-tuning)	I situasjoner der en forhåndstrent modell ikke er god nok, kan man bruke ekstra data for å spisse den til problemstillingen.
Prompt	En melding i form av tekst som blir sendt inn til store språkmodeller.
Prompt engineering	En prosess for å strukturere tekst som skal mates inn i en språkmodell. Beskriver oppgaven modellen skal gjennomføre.
Zero shot prompting	En teknikk for prompting hvor man antar at språkmodellen fra før er trent for å kunne utføre oppgaven.
Few shot prompting	En teknikk for prompting der man gir modellen eksempler på oppgaven som skal utføres.
Prompt chaining	En teknikk for prompting der man bruker utdata fra modellen videre i prompter.
Hallusinering	Et fenomen der en stor språkmodell genererer svar som ikke er grunnet i fakta.
Chunking	En metode for å dele opp tekst i flere deler, for å lettere jobbe med store mengder tekst i forbindelse med store språkmodeller.
Dokumentbit (chunk)	En del av et større dokument, som er et resultat av chunking.
Semantisk søk	Når man søker på data med lik betydning (semantikk) i stedet for enkeltord.
Vektorsøk	En metode for å utføre semantisk søk hvor tekst blir omgjort til vektorer.
Forhåndsfiltrering (Pre filtering)	Filtrering på felter i dokumentdatabase før vektorsøk utføres.
Klynge	En gruppe datapunkter som hører sammen.
K-means gruppering	En algoritme for inndeling av datapunkter i klynger.
Samhold	Et mål på hvor lik et datapunkt er til sin egen klynge.
Separasjon	Et mål på hvor lik et datapunkt er til andre klynger.
Avbildning	Oppsummering av et stykke tekst.
Reduksjon	En oppsummering av alle avbildninger.

Ord	Betydning
n-grams	Et kontinuerlig sekvens av tegn, ord eller tokener fra et dokument.
BLEU	En metode for å evaluere tekstlikhet ved hjelp av n-grams og presisjon.
BERTScore	En metode for å evaluere tekstlikhet ved hjelp av embeddinger.
Presisjon (precision)	Måler hvor ofte maskinlæringsmodellen gjør en korrekt prediksjon.
Sensitivitet (recall)	Et mål på andelen av faktiske positive tilfeller som modellen korrekt identifiserer som positive.
F1 score	Harmonisk gjennomsnitt av sensitivitet og presisjon.

Ordliste utvikling

Ord	Betydning
API (Application Programming Interface)	Grensesnitt i programvare for å kjøre kode fra annen kode.
Hosting	En metode for å gjøre nettsider og applikasjoner tilgjengelig over nettverket ved å bruke tjenere.
Webhotell	En tjeneste for å gjøre nettsider tilgjengelig over internett.
Mock	En metode hvor man setter opp data for testing, i stedet for å bruke eksisterende data. Brukes ofte for å teste mot databaser.
Nettportal	En samling ressurser og lenker på internett, innen et emneområde eller rettet mot en spesifikk brukergruppe.
LangChain	Rammeverk for å jobbe med store språkmodeller.
Pydantic	Et bibliotek for å validere data. Kan brukes sammen med LangChain for å definere modeller for utdata til store språkmodeller.
Skylagring (Blob Storage)	Fillagring i skyen.
Vektordatabase	En database som kan lagre vektorer og søke i dem.
Enhetstest	En test for å teste en mindre del av applikasjonen. Typisk en metode test.
Likhetstest	En måte å teste tekst likhet.
Ordbok (dictionary)	En Python lagringsenhet som lagrer nøkkel, verdi par.

Ordliste annet

Ord	Betydning
Kanban	En tavle som viser oppgaver som skal gjøres, er under arbeid og er ferdig.
Scrum	En arbeidsmetode for å jobbe i team, hvor arbeidsoppgaver blir delt opp i perioder.
Sprint	En periode i metoden Scrum, er typisk en eller to uker.
SUS-skjema	Evalueringsskjema for brukertester.

Akronymer

Akronym	Betydning
API	Programmeringsgrensesnitt (Application Programming Interface)
IDE	Integrert utviklingsplattform (Integrated Development Environment)
KI (AI)	Kunstig intelligens (Artificial Intelligence)
LLM	Stor språkmodell (Large Language Model)
SSA	Statens standardavtaler

INNHOLDSFORTEGNELSE

FORORD ii
Ordliste iii
1 INNLEDNING 1
1.1 Kontekst 1
1.2 Motivasjon 1
1.3 Prosjekteier 2
1.4 Problembeskrivelse og mål 2
1.5 Oppbygging av rapporten 3
2 PROSJEKTBESKRIVELSE 4
2.1 Praktisk bakgrunn 4
2.1.1 Tidligere arbeid 4
2.1.2 Initielle krav 4
2.1.3 Initiell løsnings-idé 4
2.2 Avgrensninger 5
2.3 Ressurser 5
2.4 Litteratur om problemstillingen 5
3 TEORI 7
3.1 Maskinlæring og nevrale nettverk 7
3.1.1 Trening av nevrale nettverk 8
3.1.2 Aktiveringsfunksjonen ReLU 10
3.2 Store språkmodeller 11
3.2.1 Token 11
3.2.2 Embedding 12
3.2.3 Posisjonskoding 13
3.2.4 Selv-oppmerksomhet 14
3.2.5 Transformator-arkitekturen 17
3.2.6 Forhåndstrent og finjustert modell 20
3.3 Prompt Engineering 20
3.3.1 Zero shot prompting 21
3.3.2 Few shot prompting 21
3.3.3 Prompt chaining 22

3.4 Vektordatabaser 23
4 DESIGN AV PROSJEKTET 25
4.1 Forslag til løsning 25
4.1.1 Aktuelle språkmodeller 25
4.1.2 Alternativ løsning - Trene egen språkmodell 27
4.1.3 Alternativ løsning - Ingen forhåndsbehandling 27
4.1.4 Alternativ løsning - Lokal programvare 27
4.1.5 Diskusjon av alternativene 28
4.2 Valgt løsning 29
4.3 Valg av verktøy 29
4.3.1 Programmeringsspråk, rammeverk og biblioteker 29
4.3.2 Databaser 30
4.3.3 Store språkmodeller 30
4.3.4 Utviklingsmiljø 31
4.3.5 Samarbeidsverktøy 31
4.4 Prosjektmetodikk 31
4.4.1 Utviklingsmetodikk 31
4.4.2 Prosjektplan 33
4.4.3 Risikovurdering 34
4.5 Evalueringsplan 35
4.5.1 Brukertesting 35
4.5.2 Enhetstester 35
4.5.3 Likhetstesting 36
5 DETALJERT LØSNING 37
5.1 Database og filhåndtering 37
5.1.1 Skylagring (Blob Storage) 37
5.1.2 Oppdeling av data i biter 38
5.1.3 Strukturering av databasen 38
5.1.4 Søk i database 39
5.2 Brukerhåndtering 40
5.2.1 Autentisering og autorisering 40
5.2.2 Brukerroller 40
5.2.3 Firebase Security Rules 41

5.3 Tjeneren 42
5.3.1 API og Validering av data 42
5.3.2 Service 43
5.3.3 Chains 43
5.3.4 Forretningslogikk 44
5.3.5 Asynkrone metoder 44
5.3.6 Testing 44
5.4 Klienten 45
5.4.1 Brukergrensesnitt 45
5.4.2 Ruter 46
5.5 Bruk av språkmodellene 49
5.6 Prompt Engineering 49
5.6.1 Språk 49
5.6.2 Temperatur 49
5.6.3 Tydelige instruksjoner 50
5.6.4 Formatering 51
5.7 Generering av anbud 52
5.7.1 Oppsummering av konkurranse 52
5.7.2 Oppsummering av avtaletekst 54
5.7.3 Introduksjon av bedrift 55
5.7.4 Relevante tidligere oppdrag 56
5.7.5 Valg av kompetanse 56
5.7.6 Hente ut og svare på krav for SSA-T 59
5.7.7 Beskrivelse av løsningen som skal utvikles for SSA-T 59
5.7.8 Svar på krav for SSA-T 60
6 RESULTATER 61
6.1 Evalueringsmetode 61
6.1.1 Brukertest 61
6.1.2 Testing av utdata 63
6.2 Evalueringsresultat 65
6.2.1 Brukertester 65
6.2.2 Evaluering av utdata 69
7 DISKUSJON 71

7.1 Prosjektgjennomføring 71
7.1.1 Brukertester 72
7.1.2 Tilgang på data og kompetanse 73
7.1.3 Valg av teknologi 73
7.1.4 Store språkmodeller 74
7.1.5 Refleksjon rundt valgte løsninger 74
7.1.6 Risikoutslag 75
7.2 Etiske vurderinger 76
7.2.1 Kvalitetssikring 76
7.2.2 Oppbevaring av data 76
7.2.3 Bias i utdata fra store språkmodeller 77
8 KONKLUSJON OG VIDERE ARBEID 78
8.1 Videre arbeid 79
8.1.1 Utvikling av applikasjon 79
8.1.2 Utvidet støtte 80
9 REFERANSER 81
10 VEDLEGG 84

FIGURLISTE

Figur 2-1 - Tidsbesparelser ved bruk av anskaffelsesutviklingssystem (Nistala et al., 2023, p. 14 figur 13) 6
Figur 3-1 - Lineær separerbar data (Mekeor, 2011a) 7
Figur 3-2 - Ikke-lineær separerbar data (Mekeor, 2011b) 7
Figur 3-3 - En nevron i et nevralt nettverk 8
Figur 3-4 - Nevralt nettverk med to skjulte lag 8
Figur 3-5 - Trening av nevralt nettverk. Parametere er vekt og bias, og arkitektur er modellen som skal trenes. Etiketter inneholder riktig svar for inndataene (Howard, Gugger og Chintala, 2020, s. 25 figur 1-8). 10
Figur 3-6 - ReLu aktiveringer for verdier nærme 0 (Howard, Gugger og Chintala, 2020, s. 177). 10
Figur 3-7 - "Subword tokenization" av ordet Anbud. 11
Figur 3-8 - Embedding av ordet "Anbud" (Neelakantan et al., 2022). 12
Figur 3-9 - Semantisk like embeddinger ligger nærme hverandre i vektorrommet. 13
Figur 3-10 - Posisjonskoding for fire forskjellige posisjoner (markert med prikker i ulike farger) ved bruk av sinus/cosinus-bølger (Erdem, 2021). 14
Figur 3-11 - Ord som ligger nærme gir ikke alltid konteksten som behøves. Selv-oppmerksomhet gjør at modellen kan vekte relevansen av hvert ord, selv over lengre distanser 15
Figur 3-12 - Selv-oppmerksomhet (Vaswani et al., 2017, p. 4 figur 2). 16
Figur 3-13 - Flerhodet oppmerksomhet (Vaswani et al., 2017, p. 4 figur 2) 17
Figur 3-14 - Konkatenering av oppmerksomhets-matriser fra flerhodet oppmerksomhet 17
Figur 3-15 - Residual-kobling som hopper over to lag. 19
Figur 3-16 - Transformator-arkitekturen (Vaswani et al., 2017, s. 3 figur 1) 20
Figur 3-17 - Zero shot prompt. 22
Figur 3-18 - Zero shot prompt med ukjent data, fører til hallusinering. 22
Figur 3-19 - Few shot prompt. 23
Figur 3-20 - Prompt chaining 23
Figur 3-21 - Vektorer i rom, med tre dimensjoner. 24
Figur 3-22 - En vektorspørring av ordet "kjøretøy". De nærmeste embeddingene blir returnert, blant annet embeddingen for "bil". 25
Figur 4-1 - Scrum-prosessen (Lakeworks, 2024). 33
Figur 4-2 - Kanban-brett. 33
Figur 4-3 - Oppgavetavle i tjenesten Jira. 34
Figur 5-1 - Systemarkitektur. 37
Figur 5-2 - Utsnitt av konkurranse- og konkurranseChunk-samlingene. 39
Figur 5-3 - Brukerautentisering. 40
Figur 5-4 - Firebase Security Rules. 41
Figur 5-5 - Forespørsler mot tjener. 42
Figur 5-6 - PromptTemplate 43
Figur 5-7 - Template for bruk i en PromptTemplate 43
Figur 5-8 - LangChain Expression Language (LCEL) 43
Figur 5-9 - Figma-skisse for anbudsgenerering. 45
Figur 5-10 - Ruteoversikt klient 45
Figur 5-11 - Hjemmesiden 46
Figur 5-12 - Side for anbudsgenerering 46
Figur 5-13 - Konsulentoversikt for en valgt konsulent 47
Figur 5-14 - Bedriftsdata for en bedrift 47
Figur 5-15 - Administratorside for en valgt bedrift 48
Figur 5-16 - Few shot prompting tydeliggjør oppgaven ved å gi modellen ekstra informasjon. 49

Figur 5-17 - Tydelige, positive instruksjoner brukes for å spesifisere ønsket adferd. 50
Figur 5-18 - Tydelig separasjon i prompt markerer når viktig informasjon starter og slutter. 50
Figur 5-19 - Formateringsinstruksjoner genereres fra Pydantic-klasser og settes inn i prompten 51
Figur 5-20 - Pydantic-modell med instrukser. 51
Figur 5-21 - Fargekoding av forskjellige ansvarsområder i applikasjonen 51
Figur 5-22 - Oppsummering av en anbudskonkurranse for Nord Universitet 52
Figur 5-23 - Visualisering av klynger for en anbudskonkurranse. Her med tre klynger. Vektorrommet er i denne visualiseringen redusert fra 1536 til 2 dimensjoner (Kristiansen og Vatnelid, 2024b). 53
Figur 5-24 - Flyt for å generere oppsummering av anbudskonkurranse 53
Figur 5-25 - Utsnitt fra oppsummering av avtaletekst 54
Figur 5-26 - Modell for sammendrag av avtaletekst 54
Figur 5-27 - Flyt for å hente ut viktig informasjon fra avtaletekst 54
Figur 5-28 - Flyt for å generere introduksjon av bedrift 55
Figur 5-29 - Flyt for å generere tekst om tidligere relevante prosjekter 55
Figur 5-30 - Utsnitt fra kompetansematrise 56
Figur 5-31 - Valg av konsulenter til kompetansematrise 58
Figur 5-32 - Flyt for uthenting av krav 59
Figur 5-33 - Flyt for generering av løsningsforslag 60
Figur 5-34 - Flyt for å generere svar på krav 61
Figur 6-1 - Utregning av BERTScore (Zhang et al., 2020) 65

TABELLISTE

Tabell 4-1 - Oversikt over forskjellige modeller sammen med deres priser, kontekstvindu og begrensninger på API-kall 27
Tabell 6-1 - SUS-skjema 62
Tabell 6-2 - Brukbarhet av anbudsgenerering-tjenesten 62
Tabell 6-3 - Kvalitet på utdata 63
Tabell 6-4 - Resultat av SUS-skjema etter brukertester. 67
Tabell 6-5 - Svar på spørsmål om brukbarhet av anbudsgenerering 68
Tabell 6-6 - Svar på spørsmål om kvalitet på generert tekst 68
Tabell 7-1 - Krav til applikasjonen og hvordan disse er oppfylt 70

1 INNLEDNING

1.1 Kontekst

Offentlig og privat sektor bruker *anbudskonkurranser* som en formell prosess for å kjøpe tjenester eller arbeid fra eksterne leverandører. Anbudskonkurranser kan utføres på forskjellige måter, også kjent som anskaffelsesprosedyrer. Reglene som regulerer disse anskaffelsesprosedyrene er forskjellig fra offentlig og privat sektor (Krüger og Anderssen, 2023b).

Hovedregelen i det offentlige er at kontrakter skal tildeles ved anbudskonkurranse. Offentlige innkjøp skal skje på en samfunnstjenlig måte, og med hensyn på ikke-diskriminering (Krüger og Anderssen, 2023a). *Statens Standardavtaler (SSA)* benyttes når det offentlige skal gå til innkjøp av IT og konsulenttjenester. Det finnes ulike standardavtaler for ulike tjenester, som for eksempel drift, vedlikehold, utvikling, forskning og bistand (Doffin, Database for offentlige anskaffelser, 2024c).

Mens det offentlige er pålagt å følge regelverket i lov og forskrifter om anskaffelser, kan private oppdragsgivere fritt velge hvilke regler som skal gjelde. Private oppdragsgivere er heller ikke pålagt å benytte standardavtaler, så private anbudskonkurranser kan variere vidt i innhold, format og tildelingskriterier.

Som svar på anbudskonkurranser kan leverandører avgi *anbud*, et bindende tilbud om å utføre arbeid for oppdragsgiver (Hugsted og Anderssen, 2023). Et anbud avgis på grunnlag av dokumentene i anbudskonkurransen, og kan for eksempel inneholde:

- Presentasjon av leverandøren
- Gjennomgang av krav med vekt på leverandørens tolking og forståelse av disse
- Omfattende beskrivelse av den foreslåtte løsningen
- En tabell med konsulenter som skal jobbe på prosjektet
- Eventuelt besvarelse på andre kriterier

1.2 Motivasjon

I dagens marked har leverandører en betydelig utfordring når det kommer til å delta i anbudskonkurranser. Prosessen med å skrive anbud krever mye ressurser, hovedsakelig fordi leverandørene må navigere gjennom opp til et titalls dokumenter på forskjellige filformater, med en blanding av løpende tekst og tabeller. Mye av jobben som må gjøres er å skille prosa fra faktiske krav for løsningen, og å samle alle krav og forventninger til prosjektet (Falkner *et al.*, 2019).

En besvarelse til en anbudskonkurranse krever tverrfaglig tilnærming. Bedriften har gjerne nødvendig informasjon spredt over flere interne dokumenter og systemer, noe som gjør at de må involvere mange personer i prosessen om å skrive anbud. Dette tverrfaglige behovet øker både tidsforbruket og kompleksiteten til prosessen (Falkner *et al.*, 2019).

Mye av arbeidet med anbudskonkurranser og anbud er repetitivt og arbeidsintensivt. Det ligger et stort potensial for automatisering i denne prosessen. En konkurrerende bedrift har iverksatt

en automatisert løsning for å håndtere utfordringen med anbud, og har sett betydelige besparelser i arbeidstimer. Løsningen har spart dem for millioner av kroner og tillatt dem å bruke ressurser på mer produktive oppgaver.

1.3 Prosjekteier

Link Utvikling AS er et konsulentselskap som holder til hos Bergen.Works kontorlandskap. Bedriften ble stiftet av Michelle Sæther, Hana Colakovic og Pål Vårdal Gjerde i 2018, og har i dag tre faste ansatte. De spesialiserer seg på utvikling av mobile applikasjoner og nettsider, men driver også med design og konseptualisering. Link Utvikling samarbeider ofte med NAV for opplæring og skoler for praksis og bacheloroppgaver. De er involvert i diverse frivillige organisasjoner, blant annet Oda-nettverket hvor Michelle er styremedlem. Hana er også involvert i arrangementet First Tuesday, hvor hun bidrar med å planlegge og holde foredrag på arrangementer.

Link har utviklet blant annet Kulturjakten, en mobilapplikasjon som oppfordrer til å besøke ulike kulturminner i flere byer. Prosjektet har fått støtte fra Bergen kommune og er den eneste applikasjonen som noen gang er støttet av Kulturminnefondet. De har også produsert applikasjonen Favna for B2B nettverket, som inkluderer konsept, design og utvikling av en mobilapplikasjon og en nettportal (Link Utvikling AS, 2024).

1.4 Problembeskrivelse og mål

For å forenkle prosessen med å lese anbudskonkurranser og skrive anbud, er det et ønske om å lage en *KI*-assistent som kan hjelpe leverandøren med å forstå kravene til konkurransen ved å lese inn vedlagt dokumentasjon, for så å gi en oppsummering.

Problemstillingen for oppgaven er:

> *"Hvordan kan et system basert på kunstig intelligens utvikles for å effektivisere prosessen med anbudsskriving og kompetansesøk?"*

Målet med anbudsassistenten er å lage en tjeneste som kan gjøre at bedrifter sparer tid på å lese anbudskonkurranser og skrive anbud. Dette målet kan deles opp i noen delmål, der tjenesten skal kunne:

1. *Generere oppsummeringer av anbudskonkurranser ved å laste opp vedlegg fra konkurransen.*
2. *Skrive utkast til anbud, som kan oppdateres manuelt ved behov.*
3. *Finne riktig kompetanse basert på CV-er som er lastet opp.*

Bedriften skal ha mulighet til å generere et utkast av besvarelse til konkurransen fra kravspesifikasjoner og intern informasjon, hvor det skal være mulig å enten endre utkastet direkte i tjenesten, eller laste det ned som et dokument som kan endres i for eksempel Microsoft Word.

Tjenesten skal også kunne leies ut til andre bedrifter, slik at prosjekteier kan tjene på at andre bruker den.

1.5 Oppbygging av rapporten

Rapporten er strukturert ved hjelp av IMRoD-modellen (Introduksjon, Metode, Resultat og Diskusjon).

Til nå har oppdragsgiver blitt presentert og konteksten prosjektet inngår i, samt problembeskrivelse og mål er blitt gjennomgått. I kapittel 2 blir prosjektet som skal gjennomføres beskrevet, herunder begrensninger, ressurser og krav for prosjektet. I dette kapittelet blir også relevant litteratur presentert. Kapittel 3 er en innføring i den teoretiske bakgrunnen for prosjektet, herunder maskinlæring, store språkmodeller og vektordatabaser. I kapittel 4 presenteres prosjektets design, inkludert forslag til løsning av oppgaven, diskusjon rundt mulige løsninger og endelig valg av løsning. Her gjennomgås også valg av utviklingsmetodikk og evaluering. Kapittel 5 gjennomgår den utviklede løsningen i detalj, både på klient- og tjenerside. Videre i kapittel 6 blir evalueringen av prosjektet drøftet og resultatet presentert. Kapittel 7 diskuterer konsekvenser av tilnærminger, begrensninger og verktøy, og hvilken lærdom som kan trekkes fra dette. Til slutt vil kapittel 8 drøfte om prosjektmålene er nådd, og gi forslag til videre arbeid.

Referanser og vedlegg finnes i henholdsvis kapittel 9 og kapittel 10.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

2.1.1 Tidligere arbeid

Link Utvikling har sett på lignende løsninger til konkurrerende bedrifter, Webstep og Funbit. Webstep hevder å ha spart millioner av kroner på spart arbeidstid. Link Utvikling ønsker en tilsvarende løsning som kan brukes internt i selskapet og leies ut til eksterne kunder. Det finnes også andre løsninger som er laget for det amerikanske markedet, for såkalte *RFP* eller *RFI*. Det er uvisst hvor bra disse vil virke i det norske markedet, spesielt siden RFP og RFI ikke er helt det samme som anbudskonkurranser.

2.1.2 Initielle krav

Link Utvikling har følgende krav:

> *"Lag en løsning der leverandør kan laste opp filer for å generere utkast til anbud ved hjelp av kunstig intelligens"*

Det skal utvikles en løsning der kunder kan få en bruker og logge inn for å få tilgang til systemet. Kunder skal kunne lagre egne dokumenter i systemet, som for eksempel CV-er, tidligere anbud og informasjon om bedriften. Når kunden ønsker å generere utkast til anbud, lastes alle filene til anbudskonkurransen opp i systemet, og utkastet genereres med ett klikk.

Løsningen skal kunne generere et utkast til anbud ved hjelp av *kunstig intelligens*. For å gjøre dette må man svare på kravspesifikasjon, sette opp og begrunne valg av team med nødvendig kompetanse, lage *kompetansematrise*, og skrive om bedriften og den foreslåtte løsningen. Dersom ingen konsulenter har nøyaktig den kompetansen som etterspørres, skal løsningen kunne dra sammenhenger mellom lignende teknologier for å finne den beste konsulenten for rollen.

2.1.3 Initiell løsnings-idé

Den initielle løsningsideen tar utgangspunkt i krav og ønsker fra oppdragsgiver.

Link Utvikling ønsker en løsning som kan lanseres i form av en *nettportal*. Her skal administrator for systemet kunne invitere inn bedrifter, der hver bedrift får en eller flere brukere med redigeringstillatelse. Det skal være mulig å legge inn og oppdatere kompetanse for hver ansatt i løsningen, gjerne ved å hente inn CV-er fra eksisterende systemer.

Løsningen skal ha mulighet for å laste opp flere dokumenter tilhørende en anbudskonkurranse, og lage et oversiktlig sammendrag av prosjektet og dets krav. Deretter skal det være mulighet for å generere et utkast til anbud, som tar utgangspunkt i krav funnet ved opplasting av dokumenter, tilgjengelig kompetanse som ligger lagret i systemet for bedriften, og ellers spesifikasjoner fra bedriften som gis med som en prompt.

2.2 Avgrensninger

Prosjektet vil avgrenses til å kunne generere utkast til anbud for IT-industrien. Det skal dermed ikke direkte støtte generering av anbud for andre industrier. Løsningen skal fokusere på offentlige anbudskonkurranser, siden disse følger fastsatte maler, mens private kan komme i vilkårlig format. De offentlige malene som oppdragsgiver ønsker å støtte er *SSA-B*, *SSA-B* enkel og *SSA-T*, hvor SSA-B er *bistandsavtalene* og SSA-T er *utvikling og tilpasningsavtalene* (Doffin, Database for offentlige anskaffelser, 2024c). Disse malene brukes til ulike oppdrag som legges ut på *doffin*, og et oppdrag kan bruke flere maler.

Hensikten med løsningen er ikke å generere fullstendig ferdige anbud, men å generere utkast som fungerer som et utgangspunkt for leverandør, slik at de kan spare tid på arbeidet med anbud.

2.3 Ressurser

To av gruppens medlemmer har gjennom emnet Deep Learning Engineering gjennomført et utforskende prosjekt om oppsummering av store dokumenter med kommersielt tilgjengelige språkmodeller. Resultatet av dette prosjektet er brukt som grunnlag til oppsummering av anbudskonkurranser i bachelorprosjektet (Kristiansen og Vatnelid, 2024a).

Database for offentlige anskaffelser (doffin.no) brukes for å se på anbudskonkurranser og maler. Det er også aktuelt å få tilgang til ferdige anbud som en kontroll mot resultatet fra løsningen.

For å utvikle løsningen er det behov for en *IDE* (Integrated Development Environment) for å skrive kode og tester, sammen med en form for versjonskontroll for kodeversjonering.

For å lage dette produktet skal vi ta i bruk *store språkmodeller*, som skal generere tekster og gi oppsummeringer, disse benyttes ved hjelp av et *API*.

Nettportalen krever *hosting* på et *webhotell*, dette gjelder både klient- og tjenersiden.

Gruppen skal få hjelp av veileder ved Høgskolen på Vestlandet, samt en ekstern veileder i Link Utvikling. Eventuelle kostnader til prosjektet, skal oppdragsgiver Link Utvikling bidra med. Eksempler på kostnader er tilganger til API-er og andre tjenester som trengs for å lage applikasjonen.

Generativ KI i form av ChatGPT og GitHub Copilot blir brukt i prosjektet for hjelp til feilsøking, assistanse ved koding, og innledende forklaring av konsepter som et utgangspunkt for videre lesing.

2.4 Litteratur om problemstillingen

Et litteraturstudium er gjennomført for oppgaven. Det finnes flere studier som gjennomgår lignende løsninger som den vi ønsker å produsere.

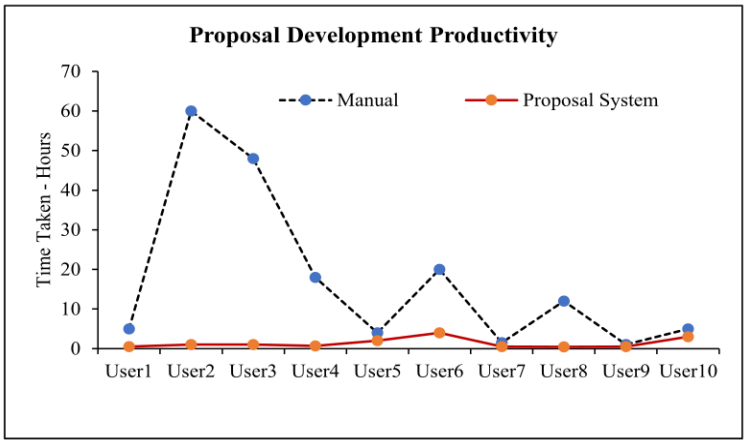
Følgende artikler handler om RFP og RFI, noe som ikke er nøyaktig det samme som en anbudskonkurranse. En RFP eller “Request for Proposal”, er en konkurranse hvor man ikke vet hvilket produkt som skal lages på forhånd. På norsk tilsvarer dette en “innovativ

anskaffelse" (Direktoratet for forvaltning og økonomistyring, 2024), og er ikke like vanlig som anbudskonkurranser. De er også mer kompliserte å skrive og tar ofte lengre tid. RFI derimot står for "Request for Information" og er når den som utlyser konkurransen trenger mer informasjon. Dette er vanligvis etterfulgt av en annen type konkurranse når man har anskaffet mer informasjon. Selv om artiklene ikke diskuterer anbud, er problemene disse forskningsrapportene utforsker likevel nærme nok problemet som skal løses i denne rapporten, til at de er relevante.

Artikkelen "An industrial experience report on model-based, AI-enabled proposal development for an RFP/RFI" (Nistala *et al.*, 2023) handler om å ta i bruk kunstig intelligens for å skrive et svar til innovative anskaffelser, de har laget en egen modell samt et verktøy for å samhandle med produktet. Dette har ført til store besparelser i tid for de fleste brukerne, hvor det beste tilfellet har gått fra rundt 60 timer til bare noen få timer. Selv om det i andre tilfeller er mindre forskjeller, så har dette ført til en god gjennomsnittlig forbedring i produktivitet (Figur 2-1).

"RFPCog: Linguistic-based Identification and Mapping of Service Requirements in Request for Proposals (RFPs) to IT Service Solutions" (Motahari-Nezhad *et al.*, 2016) forklarer problemene med å skrive svar til innovative anskaffelser, og gir forslag til en løsning ved hjelp av maskinlæringsmetoder. Hovedmålet til tjenesten *RFPCog* var å identifisere hovedtema i en innovativ anskaffelse, i tillegg til å finne alle kravene til konkurransen, og gi tilbakemeldinger på dem. Dette var den første kjente metoden for å hente og klassifisere krav fra RFP-er, ved hjelp av maskinlæring.

Maskinlæringsmodellene som blir diskutert i denne rapporten har tatt i bruk en arkitektur som benytter *selvoppmerksomhet* (self-attention). Selvoppmerksomhet slik det blir brukt i dag ble introdusert i forskningsrapporten "Attention is all you need" (Vaswani *et al.*, 2017). Rapporten og den tilhørende arkitekturen førte til enorme fremskritt innen store språkmodeller og blir sett på som fundamentet til moderne naturlig språkprosessering.



Figur 2-1 - Tidsbesparelser ved bruk av anskaffelsesutviklingssystem (Nistala et al., 2023, s. 14 figur 13)

3 TEORI

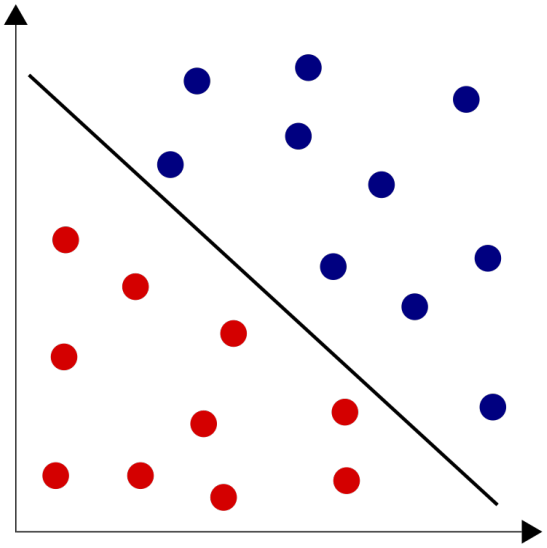
For å forstå rapporten er det nødvendig med litt kunnskap om *nevrale nettverk*, store språkmodeller og vektordatabaser. Dette kapittelet gir en innføring i teorien som trengs for å kunne lese videre i rapporten og forstå prosessen bak generativ KI.

3.1 Maskinlæring og nevrale nettverk

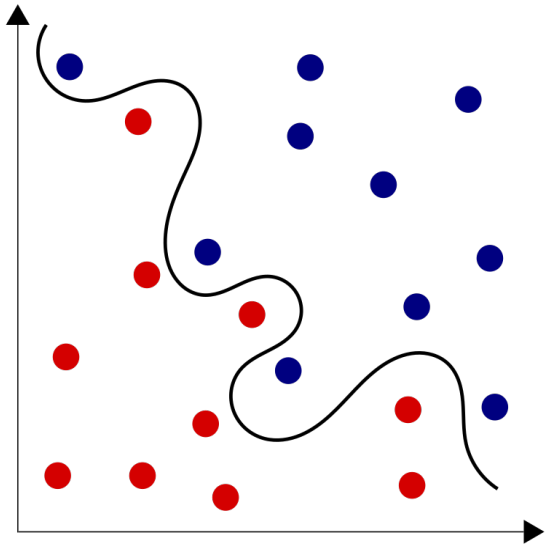
Maskinlæring er en undergren av kunstig intelligens, som handler om å trene opp modeller til å finne mønstre i data. Dette har flere ulike bruksområder, og kan for eksempel brukes for å oppdage kreft, forutse endringer i aksjemarkedet, generere tekster, lage chat-botter og mye mer.

Når man finner mønstre i data manuelt, lager vi regler som skiller data fra hverandre. En slik regel kan for eksempel være om prisen på et hus er over eller under en viss grense. Når man bruker maskinlæring, lærer modellen i stedet slike regler på egen hånd under trening. Disse reglene er gjerne ikke forståelige for mennesker, og kalles derfor ofte for svarte bokser.

Nevrale nettverk er igjen en undergren av maskinlæring, der nettverket er inspirert av menneskehjernen. Et nevralt nettverk består av kunstige nevroner, som er koblet sammen med synapser. Nevroner og synapser har tilhørende *vekter* som justeres under trening av modellen, disse påvirker styrken til signalet i en tilkobling. Nevrale nettverk lar modellen finne mer komplekse, ikke-lineære forhold i dataen (Figur 3-2), i motsetning til enklere maskinlæringsmodeller som kun finner lineære forhold (Figur 3-1).



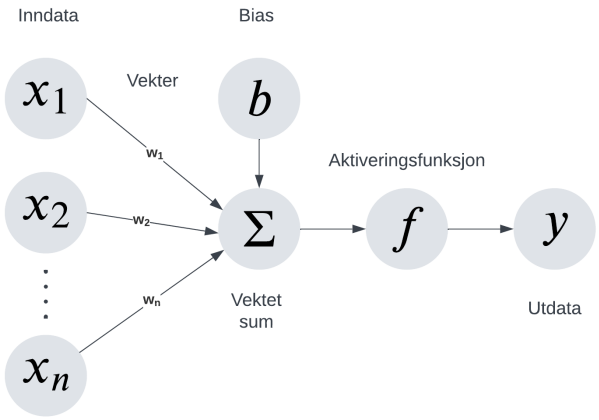
Figur 3-1 - Lineær separerbar data (Mekeor, 2011a)



Figur 3-2 - Ikke-lineær separerbar data (Mekeor, 2011b)

En kunstig nevron mottar data i form av tall multiplisert med *vekter* fra tilkoblede nevroner, legger til en *bias* (en konstant verdi lagt til summen av inndataen), prosesserer dem, og sender dem videre til de neste tilkoblede nevronene (Figur 3-3). Prosesseringen skjer i hver nevron ved hjelp av en *aktiveringsfunksjon*, en ikke-lineær funksjon som gjør det nevrale nettverket

ikke-lineært. Denne funksjonen tillater modellen å lære mer komplekse mønstre, som bildegjenkjenning og prosessering av naturlig språk.

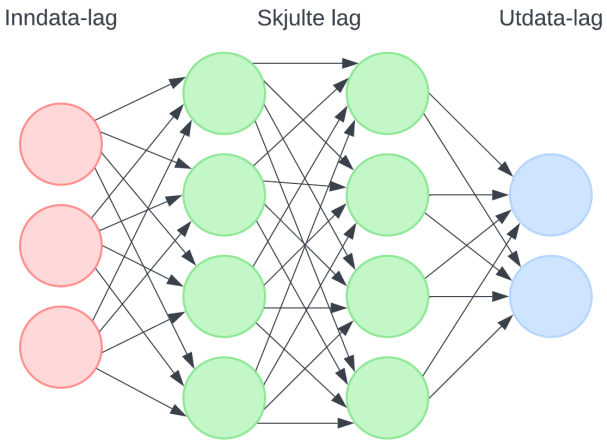


Figur 3-3 - En nevron i et nevralt nettverk

Vektet sum, den endelige inndataen til aktiveringsfunksjonen, kan beskrives slik:

$$Vektet\ sum = (x_1 * w_1) + (x_2 * w_2) + \cdots + (x_n * w_n) + b$$

Slike kunstige nevroner er typisk organisert i lag, der hvert lag utfører forskjellige transformasjoner på signalene de får inn. Signalet reiser fra inndata-laget til utdata-laget, og gjennom opptil flere mellomlag, såkalte skjulte lag (Figur 3-4).



Figur 3-4 - Nevralt nettverk med to skjulte lag

3.1.1 Trening av nevrale nettverk

Vekt er en av to trenbare parametere i nettverket og avgjør styrken på signal mellom nevroner, der større vekter betyr at inndata-nevronen har større innflytelse på utdataen. Når data sendes mellom nevroner, blir de multiplisert med denne vekten. Vekter kan være negativ eller positiv og blir initialisert med en tilfeldig verdi ved starten av trening.

Bias er den andre trenbare parameteren og er en konstant verdi som blir lagt til den summen av inndataene til nevronen. Bias kan dytte resultatet av inndataene i positiv eller negativ retning og slik ha innvirkning på aktiveringsfunksjonen i nevronen. Denne parameteren blir også initialisert med en tilfeldig verdi ved starten av trening.

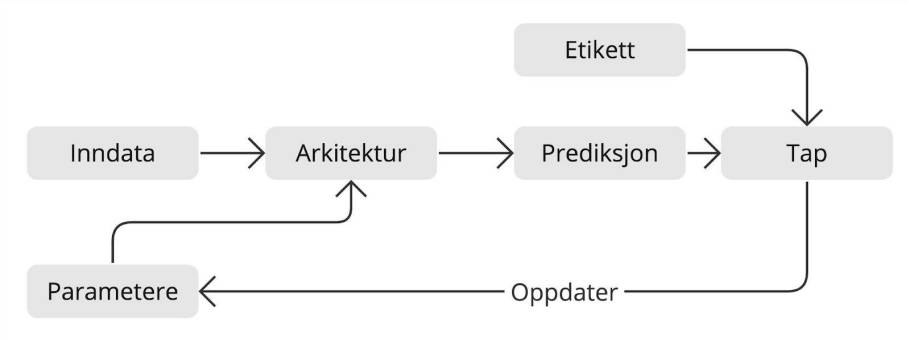
Aktiveringsfunksjonen bestemmer om nevronen skal bli aktivert, altså om den skal sende videre signalet sitt til det neste laget i nettverket.

Fremoverpropagering er det initielle steget under trening av nevrale nettverk, der inndata blir sendt gjennom det nevrale nettet for å generere en prediksjon som utdata. Utdata blir evaluert ved hjelp av en *tapsfunksjon* (loss function), som returnerer en verdi basert på en prediksjon og en faktisk *etikett* fra treningssettet, der lavere verdier tilsvarer bedre prediksjoner (Howard, Gugger og Chintala, 2020, s. 159). Det finnes mange ulike tapsfunksjoner, men *kryssentropi* (Cross Entropy) er en vanlig tapsfunksjon å bruke for store språkmodeller. Språkmodeller genererer en sannsynlighetsfordeling for alle mulige utdata, og denne sammenlignes med etiketten, altså den faktiske sannsynlighetsfordelingen for neste utdata i sekvensen, der kun sannsynligheten for faktisk utdata er 1 og resten er 0. Dersom modellen forutsier høy sannsynlighet for faktisk utdata og lav sannsynlighet for alle de andre, gir tapsfunksjonen en lav verdi.

For å forbedre det nevrale nettverket må tapsverdien minimeres, og dette gjøres ved hjelp av en optimaliseringsalgoritme kjent som *gradientnedstigning* (gradient descent). Gradienten for hver parameter måler i hvilken retning tapet øker mest, og er den partielle deriverte av tapsfunksjonen og parameterne til modellen. Gradientnedstigning fungerer ved å ta et steg i motsatt retning av gradienten for å komme frem til det minste mulige tapet. Prosessen med å regne ut gradienten for alle parametere i modellen lag for lag, kalles *bakoverpropagering*. Basert på gradienten endres så alle vekter og biaser for å minimere tapet, og prosessen gjentas (Figur 3-5).

En slik syklus med fremover og bakoverpropagering kalles en *epoke* (epoch), og treningen stopper når et forhåndsbestemt kriterie er møtt, for eksempel et gitt antall epoker.

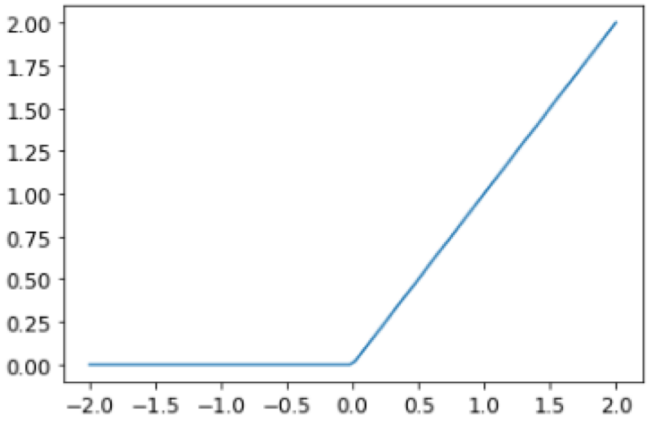
Det kan oppstå problemer med justering av vektene under trening, enten ved *eksploderende gradienter* (exploding gradients) eller *forsvinnende gradienter* (vanishing gradients), der vektene som nettverket oppdateres med blir henholdsvis større og større eller mindre og mindre etter hvert som de propageres bakover i nettverket, som kan føre til problemer med trening og forårsake dårlige tilpassede nettverk. Ved eksploderende gradienter blir de lagenes vekter oppdatert med eksponentielt større og større verdier, mens ved forsvinnende gradienter vil de første lagene i nettverket sine vekter forbli omtrent uendret da vektene blir oppdatert med mindre og mindre verdier. Eksploderende gradienter kan håndteres ved å sette en maksgrense på gradientene under bakoverpropagering, eller ved å normalisere utdataene fra hvert lag. Forsvinnende gradienter kan håndteres ved å benytte en bedre egnet aktiveringsfunksjon. Ved bruk av aktiveringsfunksjoner som presser inndataene inn i et veldig lite intervall, som for eksempel mellom 0 og 1, vil den deriverte av funksjonen nærme seg 0 når inndataene er veldig store eller veldig små, noe som fører til forsvinnende gradienter.



Figur 3-5 - Trening av nevralt nettverk. Parametere er vekt og bias, og arkitektur er modellen som skal trenes. Etiketter inneholder riktig svar for inndataene (Howard, Gugger og Chintala, 2020, s. 25 figur 1-8).

3.1.2 Aktiveringsfunksjonen ReLU

En Rectified Linear Unit (*ReLU*) er en node i et nevralt nettverk som benytter Rectified Linear Activation Function, en svært enkel ikke-lineær aktiveringsfunksjon som setter all negativ inndata til null, og beholder positiv inndata slik den er. ReLU er standard aktiveringsfunksjon i mange typer nevrale nettverk av flere grunner. Først og fremst er funksjonen svært lett å forstå og implementere, og krever lite prosesseringskraft. I tillegg fører ReLU ofte til trente modeller som yter bedre enn modeller trent med andre aktiveringsfunksjoner, da funksjonen fikser problemet med forsvinnende gradienter. Dette fordi den deriverte av ReLU er enten 0 dersom inndata er negativ, eller 1 dersom inndata er positiv, hvilket løser problemet med forsvinnende gradienter da tapet enten er 0 for noder som ikke er aktivert, eller multiplisert med 1 dersom noden er aktivert. ReLU kan derimot føre til eksploderende gradienter siden det ikke er noen begrensning på størrelsen på resultatet. Det bør derfor iverksettes tiltak for å forhindre eksploderende gradienter i nettverket, som for eksempel normalisering av utdata.



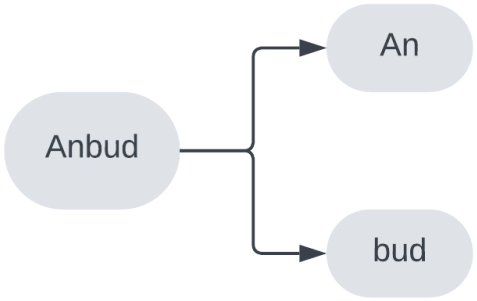
Figur 3-6 - ReLu aktiveringer for verdier nærme 0 (Howard, Gugger og Chintala, 2020, s. 177).

3.2 Store språkmodeller

Store språkmodeller (*Large Language Models, LLMs*) er en underkategori av maskinlæringsmodeller designet for å kunne forstå og generere menneskespråk. Disse modellene er bygd på nevrale nettverk for å utnytte ferdigheten slike nettverk har til å lære kompliserte mønstre og forhold mellom data. Store språkmodeller er trent på enorme mengder tekst-data, gjerne store deler av internett, noe som gjør at de presterer godt innenfor mange områder som oversetting, oppsummering, svar på spørsmål og tekstgenerering. For å forstå hvordan en stor språkmodell forstår og produserer tekst, må man først kunne litt om de grunnleggende byggeklossene til en stor språkmodell.

3.2.1 Token

En *token* er en enhet av tekst som skal prosesseres av språkmodellen. En token kan være et helt ord, ett enkelt tegn, eller en annen inndeling av teksten. Mange moderne store språkmodeller benytter "Byte Pair Encoding" (BPE), som finner en balanse mellom å benytte hele ord og enkelttegn som tokener. Dette skjer ved at man starter med et vokabular av individuelle tegn, for så å iterativt kombinere det paret med tokener som blir mest brukt sammen for å lage nye, større tokener helt til en gitt størrelse på vokabularet er nådd. *Subword tokenisering* lar modellen håndtere sjeldne og ukjente ord bedre, da slike ord kan brytes ned til kjente tokener. Selv om BPE er den mest vanlige strategien for tokenisering i store språkmodeller, vil videre eksempler i kapittelet benytte hele ord som tokener for enkelhetens skyld.



Figur 3-7 - "Subword tokenization" av ordet Anbud.

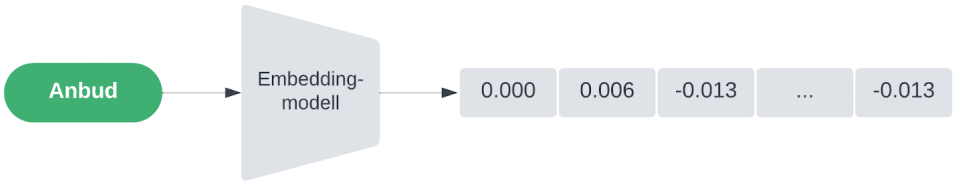
Det første som skjer i en stor språkmodell, er at inndata-teksten brytes ned til tokener. Store språkmodeller har begrensninger på hvor mange tokener de kan prosessere på en gang, både som inndata og utdata. Dette kalles *kontekstvinduet* til modellen. Størrelsen på dette kontekstvinduet varierer fra modell til modell, men inkluderer alltid tokener og metadata. Videre blir den inndelte teksten matet inn i en embedding-modell.

3.2.2 Embedding

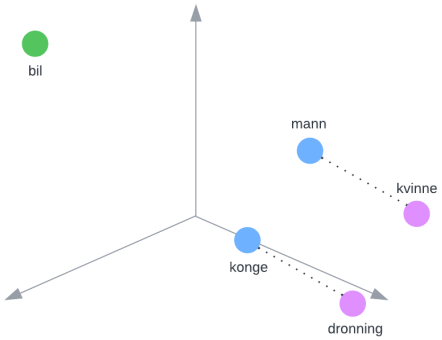
Siden nevrale nettverk opererer med tall, og store språkmodeller ønsker å jobbe med tekst, må inndata gjøres om fra tekst til tall. For å oppnå dette brukes en prosess kalt *embedding*, der tekst sendes gjennom et relativt simpelt nevralt nettverk, en embedding-modell, for å gjøres om til en vektor (Figur 3-8).

Slike embedding-modeller kan trenes på forskjellige måter, for eksempel ved å gjette ordene som skal være før og etter et gitt ord, eller ved å gjette et ord fra konteksten rundt. Under trening justeres embedding-vektorene for å maksimere sannsynligheten for å finne riktig ord. Som en konsekvens av dette vil vektorene bli justert slik at ord som ofte dukker opp i lignende sammenhenger er nærmere hverandre, og ord som ikke dukker opp i samme kontekst dyttes fra hverandre. Over mange iterasjoner, og med store datasett, vil modellen kunne fange opp semantisk likhet (Figur 3-9), men også syntaks, fordi ord med samme ordklasse opptrer i lignende kontekster. Mer kompliserte modeller kan lage gode embeddinger for lengre sekvenser av tekst, med mange inndata-tokener.

Grafikkprosessorer (GPUer) prosesserer embedding-vektorer mest effektivt når de passer sammen med antall tråder i en warp, en prosesseringsenhet der hver tråd utfører samme operasjon på forskjellig data. Disse prosesseringsenhetene har ofte 32 tråder, og embedding-modeller produserer derfor ofte vektorer med dimensjoner som går opp i 32 for å utnytte alle trådene. Embedding-modellen som er brukt i dette prosjektet er Text Embedding 3 Small, som produserer embeddinger med en dimensjon på 1536, som tilsvarer 48 * 32.



Figur 3-8 - Embedding av ordet "Anbud" (Neelakantan et al., 2022).



Figur 3-9 - Semantisk like embeddinger ligger nærme hverandre i vektorrommet.

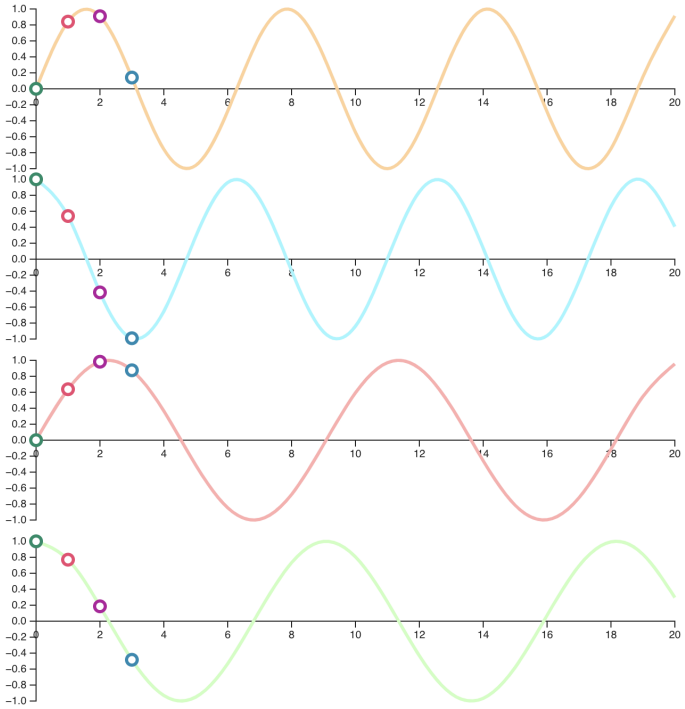
3.2.3 Posisjonskoding

Siden ord kan endre betydning ut fra plassering og kontekst, er det ønskelig å inkludere informasjon om posisjonen til en token i inndataen. Dette gjøres ved *posisjonskoding*, en prosess der hver mulig posisjon i en tekst har sin egen forhåndsdefinerte vektor. En vanlig måte å generere unike vektorer for hver posisjon på, er en rekke med sinus og cosinus-funksjoner, der hver posisjon skjærer sinus og cosinus-bølgene på et eget punkt. Bølgene gjentas, men med forskjellige intervaller, og kombinasjonen av disse fører dermed til unike posisjonskodinger for hver posisjon i teksten (Figur 3-10). Den posisjonelle vektoren og embeddingen på tilsvarende plass blir summert sammen med vektoraddisjon for å danne en ny vektor som inneholder informasjon om posisjon i teksten. Under er et enkelt eksempel på posisjonskoding av en vektor på posisjon 0, ut fra posisjonsvektorene fra Figur 3-10:

$$embedding_0 = [1, 2, 3, 4]$$

$$posvektor_0 = [0, 1, 0, 1]$$

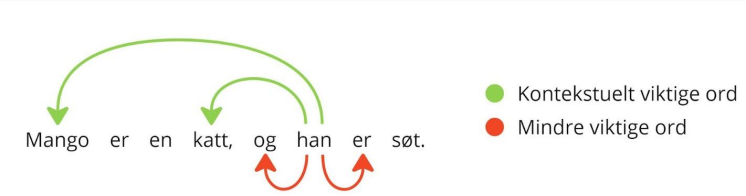
$$poskoding_0 = [(1 + 0), (2 + 1), (3 + 0), (4 + 1)] = [1, 3, 3, 5]$$



Figur 3-10 - Posisjonskoding for fire forskjellige posisjoner (markert med prikker i ulike farger) ved bruk av sinus/cosinus-bølger (Erdem, 2021).

3.2.4 Selv-oppmerksomhet

Problemet med modeller som ikke benytter oppmerksomhet er at det er vanskelig for disse modellene å dra sammenhenger over lange distanser i teksten, og å finne den mest relevante delen av teksten for å generere neste token. Oppmerksomhet er en mekanisme som løser problemet med kontekst i språkmodeller, ved å tillate modellen å vekte viktigheten av hvert ord i en tekst i relasjon til hverandre.



Figur 3-11 - Ord som ligger nærme gir ikke alltid konteksten som behøves. Selv-oppmerksomhet gjør at modellen kan vekte relevansen av hvert ord, selv over lengre distanser

Ved å vekte hver token-embedding i inndataene, vil den resulterende embeddingen inneholde mer kontekst enn den originale embeddingen. Siden semantisk like ord har embeddinger som ligger nærmere hverandre i vektorrommet, kan man utnytte dette til å tilføre embeddingene mer kontekst.

Selv-oppmerksomhet har tre forskjellige inndata, kalt *spørring*, *nøkler* og *verdier*. Selv om de har forskjellig navn, er alle disse inndataene embeddinger fra teksten. Spørring er den aktive embeddingen, den vi ønsker å tilføre kontekst. Nøkler og verdier er rett og slett alle embeddinger fra teksten. Grunnen til at disse inndataene har forskjellig navn er på grunn av trenbare matriser, som vil forklares senere. Nøkler brukes for å vekte viktigheten av hver embedding i forhold til spørringen. Verdier brukes for å finne den endelige, vektede embeddingen for hver token i teksten.

For å finne vekten (w) for den første tokenen i teksten (*spørring*), finnes indreproduktet av den første tokenen og alle tokener i teksten inkludert den første tokenen (*nøkler*), slik at man ender opp med en vekt per token-par. Siden vektorer med større dimensjon vil gi større varians i indreprodukt, kan dette føre til svært store eller svært små vekter. Dette kan føre til feilaktig fordeling i neste lag. Derfor blir disse vektene skalert ved å dele dem med kvadratroten av dimensjonen til spørring-vektoren. På slik måte reduseres variansen introdusert av dimensjonen til vektoren.

$$spørring \cdot nøkkel_i = x_i$$

$$w_i = \frac{x_i}{\sqrt{d_{spørring}}}$$

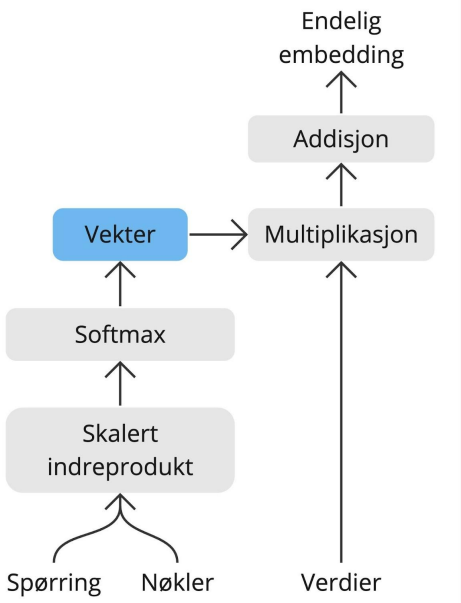
Den resulterende vekten kan være positiv, negativ, og kan eksistere utenfor intervallet (0,1). For å finne ut hvor viktig hver token er i forhold til spørring-tokenen, altså hvilken vekt som skal ha mest innvirkning på den resulterende vektoren, må det lages en fordeling der alle vektene eksisterer i intervallet (0,1) og summerer til 1, der rekkefølgen på vektene er bevart fra lav til høy. Dette gjøres ved hjelp av Softmax-laget. Man kan tenke på dette laget som en måte

å finne ut hvor stor prosentandel av hver nøkkel man skal bruke for å kode spørringen, der nøkler med høyere vekt har større innvirkning. Videre blir disse "prosentandelene" for hver nøkkel multiplisert med den originale embeddingen til hver token i teksten (*verdier*), og deretter summert, og slik ender man opp med den endelige embeddingen. Under vises utregning av oppmerksomhet (o) for hver token i teksten, og den endelige, konstekstrike embeddingen (e) for spørringen.

$$o_i = softmax(w_i) * verdi_i$$

$$e_{spørring} = o_1 + o_2 + \cdots + o_n$$

Denne endelige embeddingen inneholder kontekst fra hver annen token inkludert seg selv i teksten, der mer relevante tokener har større innflytelse på embeddingen (Figur 3-12). Denne prosessen gjentas for alle tokener i teksten.



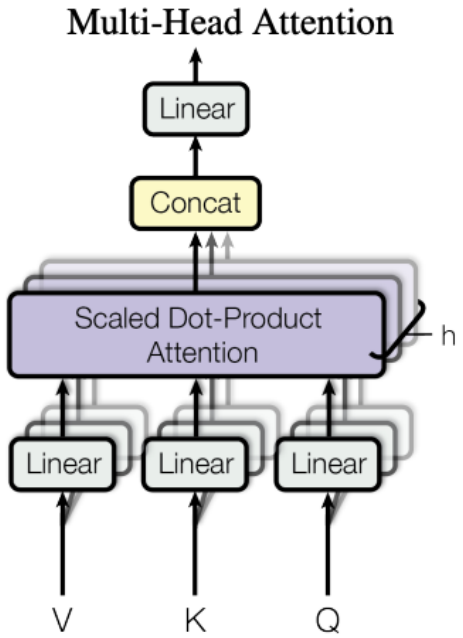
Figur 3-12 - Selv-oppmerksomhet (Vaswani et al., 2017, s. 4 figur 2).

En strategi for å tilføre enda mer kontekst er å legge til parametere som kan trenes. Disse parameterne finner vi i form av tre forskjellige matriser, som hver er et sett med vekter optimalisert under trening av modellen. De tre matrisene hører henholdsvis til spørring, nøkler og verdier, og multipliseres med henholdsvis spørring-vektoren, nøkkel-vektorene og verdi-vektorene for å gi mer kontekst til hver vektor.

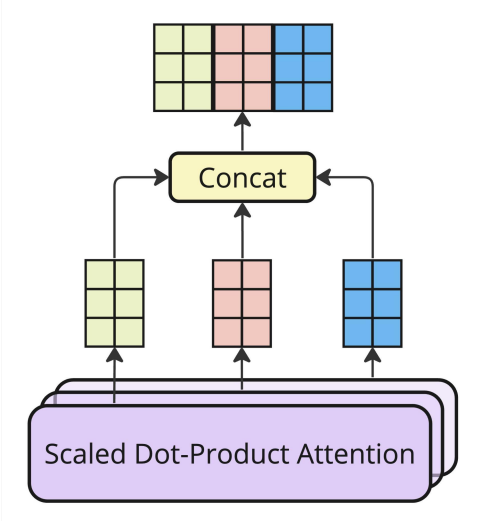
For å oppsummere: Embeddingene går først gjennom lineære lag som kun inneholder matriseregning, der resultatene blir kalt spørring, nøkler, og verdier. Indreproduktene av spørring og nøkler blir skalert for å finne vektene, og så normalisert ved et softmax-lag. "Prosentandelene" fra softmax-laget blir multiplisert med verdier, og summert for å finne den endelige oppmerksomhetsvektoren. Hele denne prosessen kalles et oppmerksomhets-lag, og flere slike kan stables for å danne *flerhodet oppmerksomhet* (Multi-Head Attention).

Flerhodet oppmerksomhet (Figur 3-13) lar hvert “hode” (oppmerksomhets-lag) fokusere på ulike kvaliteter ved teksten, for å få en enda dypere forståelse for konteksten. Derfor bruker

hvert lag hver sin del av de trente matrisene for spørringer, nøkler og verdier. Oppmerksomhets-lagene i flerhodet oppmerksomhet er mindre enn ett enkelt oppmerksomhets-lag ville vært, for å redusere ressursbehovet. De kan kjøre parallelt med hverandre og tillater dermed bruk av flere kjerner. Resultatet av flerhodet oppmerksomhets-laget er resultatet fra alle oppmerksomhets-lagene konkatenert (Figur 3-14). For å unngå at resultatet blir større og større desto flere hoder vi har, sendes resultatet gjennom en lineær funksjon som reduserer dimensjonen til matrisen ned til den originale embedding-størrelsen.



Figur 3-13 - Flerhodet oppmerksomhet (Vaswani et al., 2017, s. 4 figur 2)



Figur 3-14 - Konkatenering av oppmerksomhets-matriser fra flerhodet oppmerksomhet

3.2.5 Transformator-arkitekturen

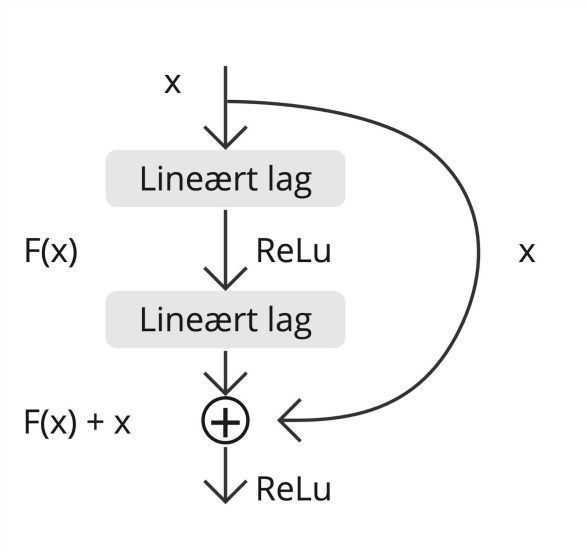
Med kunnskap om tokener, embedding, posisjonskoding og selv-oppmerksomhet kan man ta fatt på *transformator-arkitekturen*, for å lære om hvordan en stor språkmodell som bruker denne arkitekturen genererer tekst. En visuell representasjon av arkitekturen finnes i Figur 3-16. Dette kapittelet beskriver transformator-arkitekturen slik den er satt opp i "Attention is All You Need" (Vaswani *et al.*, 2017).

Arkitekturen er satt opp med stabler av koding og dekoding-lag. Koding-lagene er ansvarlig for å avbilde inndatasekvensen til en kontekstrik vektor-representasjon som kan benyttes til generering av utdata. Dekoding-lagene genererer utdatasekvensen en token av gangen, ved å bruke vektor-representasjonene fra koding-lagene. Lagene inneholder tre forskjellige oppmerksomhets-komponenter:

- Selvoppmerksomhet i koding-lag: Inndatasekvensen har oppmerksomhet mot seg selv.
- Maskert selvoppmerksomhet i dekoding-lag: Utdatasekvensen har oppmerksomhet mot seg selv.
- Koding-dekoding-oppmerksomhet i dekoding-lag: Utdatasekvensen har oppmerksomhet mot inndata-sekvensen.

Hvert koding- og dekoding-lag inneholder også et *fremovermating-lag*. Fremovermating-lagene er fullt tilkoblede nevrale nettverk der dataflyten skjer i kun én retning. Disse nettverkene består av to lineære transformasjoner med en ReLU-aktivering mellom. De lineære transformasjonene øker dimensjonaliteten til inndataene for å mate en rikere representasjon av dataene til aktiveringsfunksjonen, for så å krympe dimensjonaliteten ned til original størrelse igjen etter aktiveringsfunksjonen. Transformasjonene utføres individuelt, men likt, på hver posisjon i teksten.

Etter hvert oppmerksomhet og fremovermating-lag i koding og dekoding-lagene, sendes resultatet inn i en residual-kobling sammen med dataen slik den var før den kom inn i det aktuelle laget. *Residual-koblinger* (Residual Connection) er en teknikk for å forenkle treningen av svært dype nevrale nettverk ved å tillate data å hoppe over lag i nettverket. Slike lag gjør det lettere for dataene å konvergere, altså å finne stabile verdier for vekter og bias som produserer gode utdata.

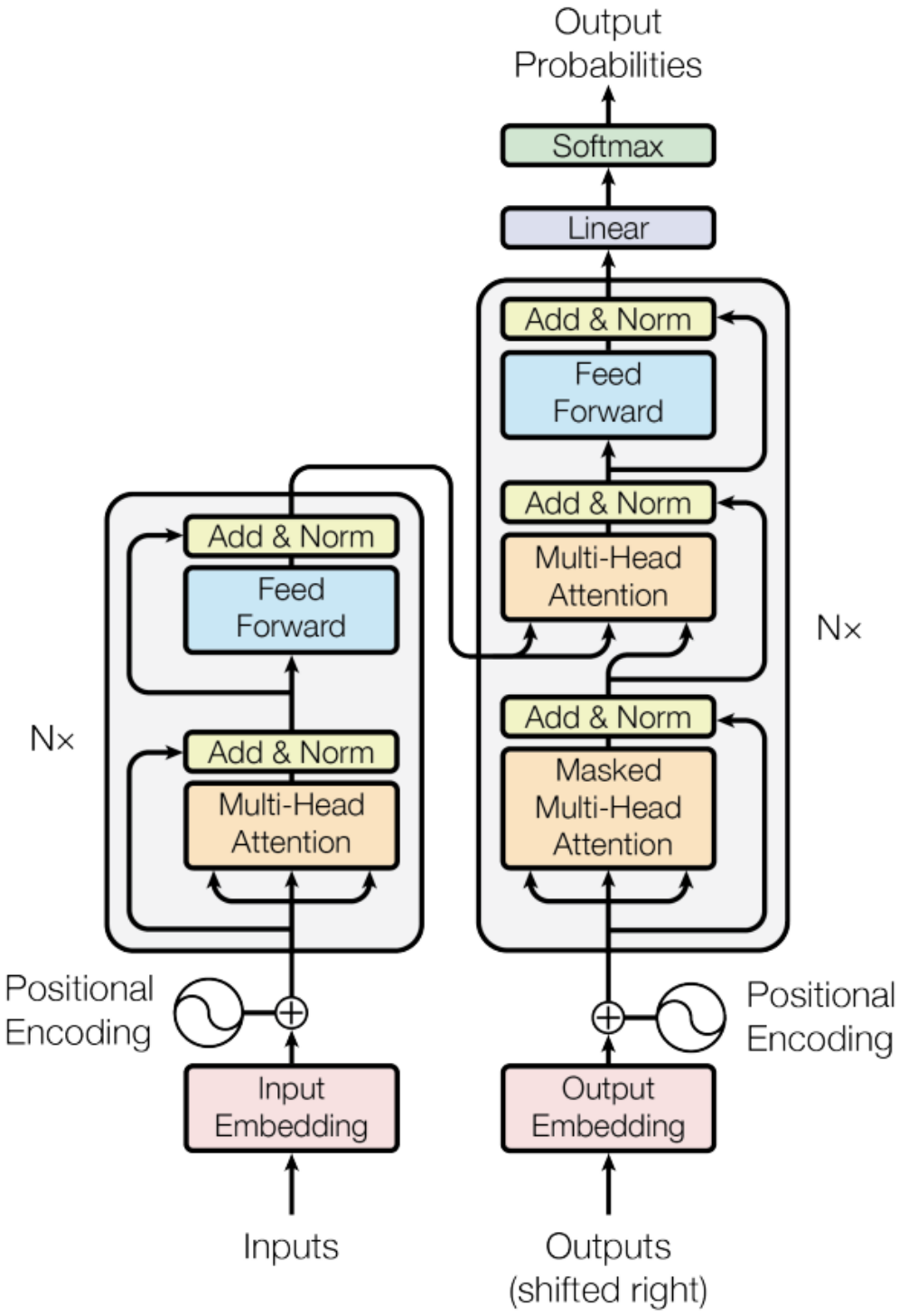


Figur 3-15 - Residual-kobling som hopper over to lag.

Koding-lagene inneholder selv-oppmerksomhetslag der alle spørringer, nøkler og verdier kommer fra samme plass, altså forrige lag i koding-laget.

Dekoding-lagene inneholder to forskjellige oppmerksomhetslag. Det første er et maskert selv-oppmerksomhetslag der alle posisjoner som kommer etter aktiv posisjon er maskert, altså blir de satt til -∞ i oppmerksomhetslagets softmax-funksjon. Dette gjøres for å sørge for at dekoderen er auto-regressiv, at den kun benytter tidligere data for å forutsi neste utdata under trening. Det andre oppmerksomhetslaget er koding-dekoding-laget, der spørringene kommer fra forrige dekoding-lag, mens nøkler og verdier kommer fra utdataen fra koding-laget. Utdataen fra dekoder-lagene er en vektor med poengsummer per mulige token, der dimensjonen på vektoren er lik vokabularet (alle kjente tokener) til modellen. Høyere poengsum betyr at modellen er mer sikker på at denne tokenen burde være neste i sekvensen.

Token med høyest poengsum kan bli valgt som utdata, eller man kan innføre en viss mengde tilfeldighet i valg av token ved bruk av en parameter kalt *temperatur*. En token blir valgt, med eller uten tilfeldighet, og blir lagt til utdataen. Utdataene sendes gjennom dekoder-lagene igjen og prosessen gjentas helt til en End of Sequence-karakter (<EOS>) blir valgt.



Figur 3-16 - Transformator-arkitekturen (Vaswani et al., 2017, s. 3 figur 1)

3.2.6 Forhåndstrent og finjustert modell

Store språkmodeller lærer ikke å utføre noen spesifikke oppgaver under trening. I stedet lærer den å generere tekst ved å forutsi neste ord ut fra konteksten. Den generelle forståelsen for språk som modellen sitter igjen med etter trening er grunnlaget for å kunne løse et bredt spekter av problemer, som oppsummering, oversetting, tekstgenerering og svar på spørsmål. Etter denne initielle treningen ender man opp med en *forhåndstrent* modell.

Som et eksempel kan modellen finne adjektiver i en tekst, selv om modellen ikke spesifikt har blitt trent til denne oppgaven. Under trening har modellen blitt eksponert til utallige setninger der adjektiver har blitt brukt for å beskrive substantiver. Dermed har modellen lært ordbruk og posisjonering som er karakteristisk for adjektiver, altså mønstergjenkjenning. Selv-oppmerksomhet gjør at modellen kan evaluere konteksten rundt et ord. Når modellen så blir bedt om å finne eller generere adjektiver, brukes denne konteksten for å velge et ord som grammatisk og semantisk passer til å være adjektiver. Treningen inkluderer eksempler på adjektiver brukt i et vidt spekter av sammenhenger. Dermed forstår modellen at adjektiver betegner egenskaper ved substantiver, uavhengig av den spesifikke setningen den genererer eller analyserer.

Det finnes derimot oppgaver modellen ikke kan løse uten videre. Et eksempel på dette er domenespesifikk informasjon som modellen ikke har blitt eksponert for under trening, som interne bedriftsdokumenter når man lager en chatbot for kundeservice. For å kunne generere svar ut fra disse dokumentene, må modellen gjennom en prosess som heter *finjustering.* Under finjustering mates modellen med dokumentene den trenger for å løse domenespesifikke oppgaver, som å gi informasjon om retur-retningslinjer for en bedrift, og deretter oppdateres alle de interne vektene i det nevrale nettverket for å tilpasses den nye informasjonen. Den finjusterte modellen kan så generere svar ut fra den nye dataen den har blitt gitt.

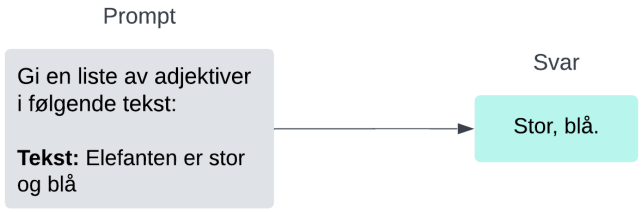
3.3 Prompt Engineering

For å kunne effektivt kommunisere med store språkmodeller må man bruke *prompt engineering*. *Prompter* er en måte å kommunisere med store språkmodeller i form av spørsmål og informasjon. En prompt påvirker interaksjoner med, og utdata fra, store språkmodeller. Riktige prompter utgjør stor forskjell på resultatet fra språkmodellen, og dermed brukbarheten til resultatet. Derfor er det gjort forsøk på å lage en systematisk tilnærming til prompting, nemlig prompt engineering.

Innenfor disiplinen prompt engineering, er det et utvalg teknikker man kan bruke for å optimalisere prompter. Teknikkene "*zero shot prompting"*, "*few shot prompting"* og "*prompt chaining"* gjennomgås i dette kapittelet.

3.3.1 Zero shot prompting

I en zero shot prompt er målet å få modellen til å generere et svar uten å sende med ekstra informasjon. Utgangspunktet til zero shot prompting er at modellen allerede er trent på store mengder data og kan utføre et bredt spekter av oppgaver uten ekstra trening (Dair.AI, 2024c). Figur 3-17 viser et eksempel på en slik prompt, der modellen fra trening har blitt eksponert til grammatiske strukturer og forhold, og kan gjenkjenne et adjektiv ut fra kontekst og posisjon i teksten.

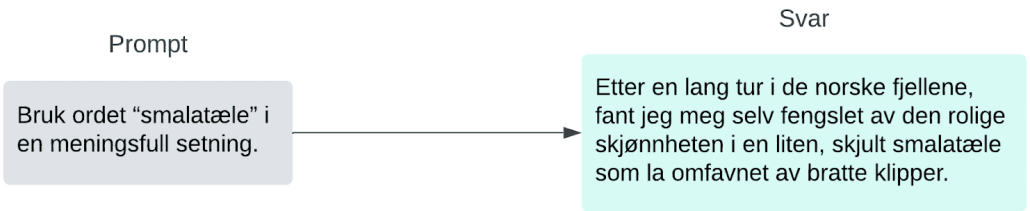


Figur 3-17 - Zero shot prompt.

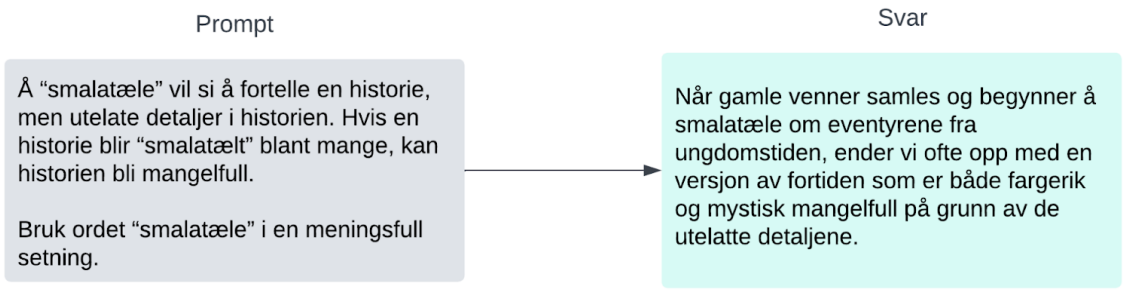
3.3.2 Few shot prompting

Ved few shot prompting gir man modellen ekstra informasjon om ønsket resultat. Det kan være for eksempel definisjonen på ukjente ord eller eksempel på forventet resultat (Dair.AI, 2024a). Dette er et naturlig steg å ta dersom man har prøvd en zero shot prompt, og ikke får et svar man er fornøyd med.

Når man ber en modell utføre oppgaver den ikke kan noe om, er det en fare for et fenomen som kalles *hallusinering,* der modellen genererer svar som ikke er basert på realiteten (Figur 3-18). For å unngå hallusinering, bør man gi modellen nødvendig informasjon når man utfører oppgaver som modellen ikke er trent på. Dersom modellen blir supplert med informasjonen den trenger for å gi et riktig svar på prompten, kan den generere tekst som ikke er en hallusinasjon (Figur 3-19).



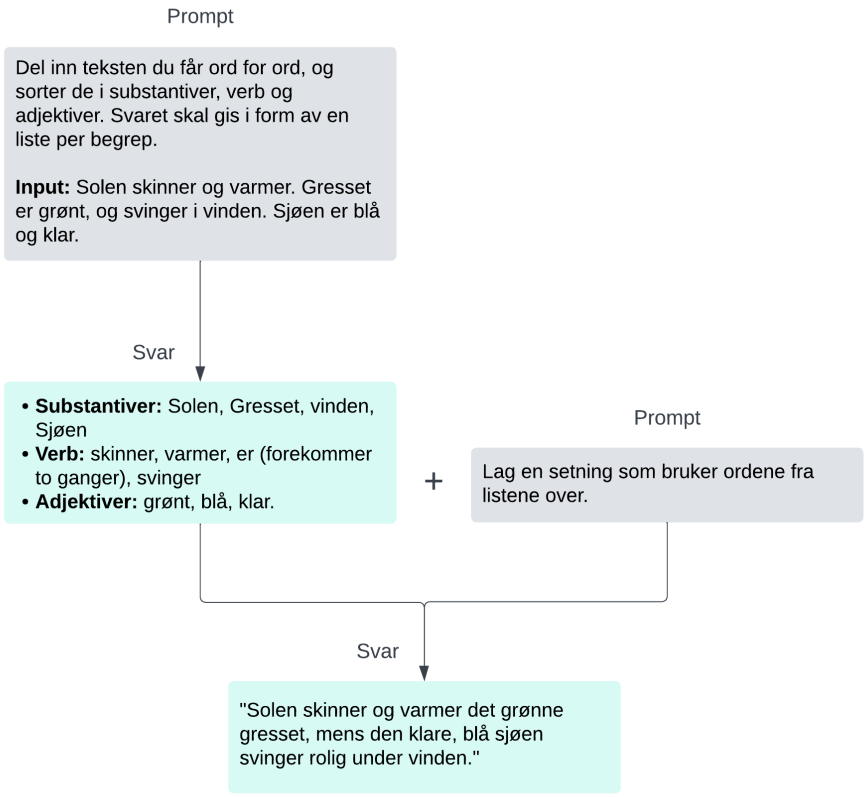
Figur 3-18 - Zero shot prompt med ukjent data, fører til hallusinering.



Figur 3-19 - Few shot prompt.

3.3.3 Prompt chaining

Ved å benytte prompt chaining kan man bruke svaret fra språkmodellen i en ny prompt, og på denne måten "lenke" prompter sammen. Prompt chaining er nyttig blant annet når man vil hente ut informasjon fra et dokument, og så generere ny tekst ut fra informasjonen som ble hentet (Dair.AI, 2024b). Et enkelt eksempel på prompt chaining kan sees i Figur 3-20.



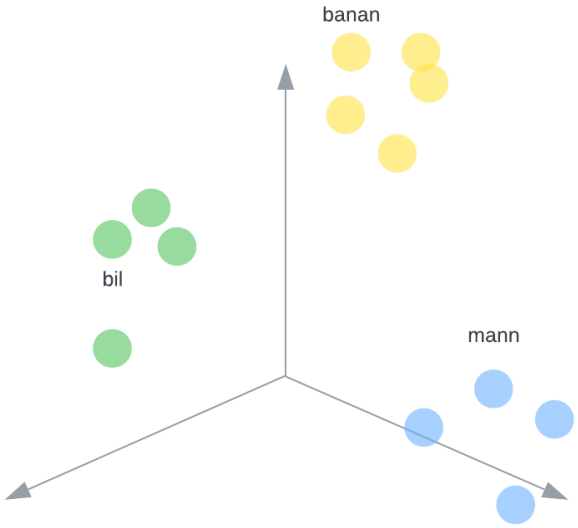
Figur 3-20 - Prompt chaining

3.4 Vektordatabaser

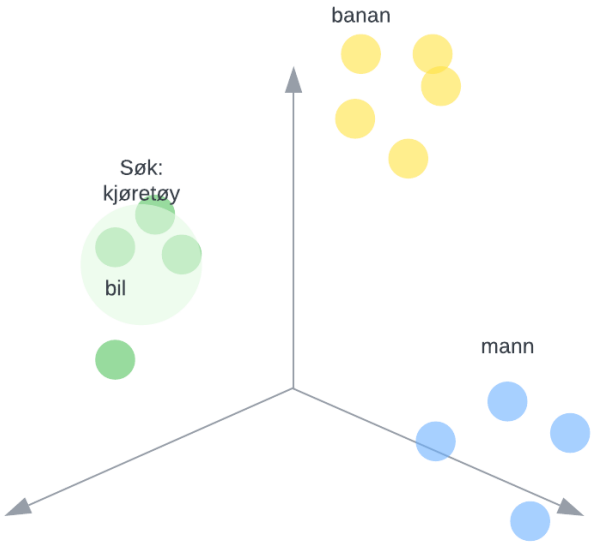
Vektordatabaser lagrer data som vektorer, altså numeriske representasjoner av flerdimensjonale data. Hvor mange dimensjoner en slik vektor har, varierer ut fra hvilken embedding-modell eller metode som brukes for å lage vektorene. Hver vektor kan sees på som et punkt i et flerdimensjonalt rom, der relaterte punkter ligger nær hverandre i rommet (Figur 3-21).

Spørringer mot en vektordatabase skjer med en vektor eller et sett med vektorer, der man får tilbake dataene som ligger i de nærmeste vektorene i rommet (Figur 3-22). Dette kalles et *vektorsøk*. I prosjekter som benytter embeddinger vil en spørring bli gjort i klartekst, som mates inn i samme embedding-modell som er brukt til å generere embeddingene som ligger i databasen, og den resulterende embedding-vektoren brukes så til spørring mot databasen. Denne søkevektoren plasseres midlertidig i vektorrommet. Ut fra denne vektoren beregnes avstanden til de andre vektorene i rommet med cosinus-likhet, som beregner likheten mellom to vektorer ut fra hvor stor vinkelen mellom vektorene er, der mindre vinkel tilsier større likhet. De *n* likeste vektorene velges, hvor *n* representerer det antallet vektorer man søker etter.

Vektordatabaser blir mye brukt til prosjekter som benytter seg av embedding-baserte maskinlæringsmodeller da det gjør det mulig å utføre *semantisk søk*. Et eksempel er språkbehandlingsapplikasjoner der det er behov for å hente ut relevante dokumenter eller svar basert på semantisk likhet i stedet for nøkkelord-søk, blant annet chatboter for nettsteder. Bedriften kan da legge inn dokumenter spesifikk til nettstedet, produktet og kundebehandling i en vektordatabase, og chatboten vil da generere svar ut fra de mest relevante dokumentene til en bruker sitt spørsmål.



Figur 3-21 - Vektorer i rom, med tre dimensjoner.



Figur 3-22 - En vektorspørring av ordet "kjøretøy". De nærmeste embeddingene blir returnert, blant annet embeddingen for "bil".

4 DESIGN AV PROSJEKTET

4.1 Forslag til løsning

Det initielle løsningsforslaget er en KI-basert applikasjon lansert i en nettportal. Nettportalen benytter seg av et brukersystem slik at kun de som har kjøpt tilgang kan bruke applikasjonen. Bedrifter som har kjøpt tilgang til applikasjonen får mulighet til å legge inn brukere under bedriftens konto, samt laste opp interne dokumenter som CV-er og informasjon om bedriften. Hver bedrift har egne dokumenter lagret i systemet og kan se sine tidligere genererte anbud i nettportalen.

Applikasjonen er tenkt bygd med en klient-tjener-arkitektur, der klienten håndterer det grafiske grensesnittet og brukerinteraksjonen. Tjeneren vil håndtere den logiske delen av applikasjonen, som formatering av data, kommunikasjon med API-er og database. Applikasjonen benytter også to forskjellige typer databaser, en dokumentdatabase for lagring av brukerinformasjon og dokumenter, samt en vektordatabase for lagring av og søk i embeddinger.

Løsningen benytter en forhåndstrent språkmodell som kan kommuniseres med via et API. Dataene som sendes inn til språkmodellen forhåndsbehandles på tjenersiden før innsending. Dette gjør dataene lettere å lese for språkmodellen, og data kan deles opp i mindre biter for å sende med kun relevant informasjon til modellen.

4.1.1 Aktuelle språkmodeller

Kapittelet gjennomgår noen kjente kommersielt tilgjengelige store språkmodeller som kan være aktuelle for prosjektet. Til slutt vil alle modellene sammenlignes i Tabell 4-1.

OpenAI

OpenAI sitt API gir tilgang til OpenAI sine store språkmodeller, samt funksjonalitet for å jobbe med dem. Den siste stabile modellen er GPT-4, men GPT-4 Turbo er en nyere modell som ved starten av prosjektet er i *offentlig beta*. Fordelen med GPT-4 Turbo er at den er raskere enn GPT-4, og har et mye større kontekstvindu. Det kan også være aktuelt å bruke GPT-3.5 modellen til visse ting, da ikke alle oppgaver som skal løses er kompliserte, og GPT-3.5 er raskere og mye billigere til slike enkle oppgaver (OpenAI, 2024c).

API-et har også støtte for å laste opp filer, hvor en modell kan brukes for å lese gjennom innholdet i filen, for eksempel for å gi oppsummeringer på innholdet eller hente ut info. Dette kan gjøres ved bruk av Assistants API-et. Dette API-et tilbyr også annen funksjonalitet, blant annet muligheten til å lage tråder for å holde en samtale med modellen. Assistants API-et er foreløpig i *offentlig beta,* så endringer kan skje underveis (OpenAI, 2024a).

Alternativet til å bruke Assistants API-et er å implementere funksjonalitet på egen hånd ved hjelp av andre biblioteker. Det er mulig å bruke biblioteker for å lese filer til klar tekst, før teksten blir matet til GPT-modellen. Tråder kan også implementeres manuelt ved å ta vare på gamle meldinger, og sende dem med til modellen i neste melding, som en del av prompten.

Anthropic Claude

Anthropic har i løpet av prosjektets periode sluppet tre nye modeller, Claude Opus, Sonnet og Haiku. Disse modellene har mye større kontekstvindu enn OpenAI sine GPT-modeller (Anthropic, 2024a), men har svært stramme begrensninger på antall API-kall og mengde tokener man er tillatt å sende (Anthropic, 2024b), i tillegg til at mengden utdata er det samme som OpenAI sine modeller.

Llama 2

Meta AI har sin egen serie med store språkmodeller som heter Llama. Ved prosjektets oppstart var Meta AI sin nyeste modell Llama 2. Modellen har åpen kildekode og kan kjøres på egen tjener. Dette gjør at produktet kan bli billigere, da man bare må betale for å hoste modellen på tjeneren. Imidlertid kan dette føre til at man må sette opp en del infrastruktur, siden den beste Llama 2 modellen krever mye minne (Umbrel, 2024). Den har også mye mindre kontekstvindu enn alle andre modeller gjennomgått i dette kapittelet.

Tabell 4-1 - Oversikt over forskjellige modeller sammen med deres priser, kontekstvindu og begrensninger på API-kall

Modell	**Pris per 1m inndata-tokens**	**Pris per 1m utdata-tokens**	**Kontekstvindu**	**Forespørsler i minuttet**	**Tokens i minuttet**
OpenAI (OpenAI, 2024c, 2024f)					
GPT-3.5 Turbo	$0.50	$1.50	16 000	Opp til 10 000	2 millioner
GPT-4 Turbo	$10	$30	128 000	Opp til 10 000	1,5 millioner
Anthropic Claude (Anthropic, 2024a, 2024b)					
Haiku	$0.25	$1.25	200 000	Opp til 4000	Opp til 400 000
Sonnet	$3	$15	200 000	Opp til 4000	Opp til 400 000
Opus	$15	$75	200 000	Opp til 4000	Opp til 400 000
Meta AI (Meta, 2024)					
Llama 2	Ingen	Ingen	4000	Ingen begrensning	Ingen begrensning

4.1.2 Alternativ løsning - Trene egen språkmodell

Et alternativ er å trene opp en egen språkmodell som vi kunne brukt til å generere anbudene med. Dette ville gitt prosjektgruppen større kontroll over hva modellen blir trent på, og dermed også kunne spisse svarene vi får. Dette er også en løsning som vil være mindre økonomisk kostbar i lengden enn å bruke kommersielle språkmodeller.

Det er betydelige utfordringer med å trene en egen språkmodell som vil kunne konkurrere med forhåndstrente modeller. En stor utfordring er mengden data som trengs for å trene modellen. Her er det behov for store mengder av både anbudskonkurranser og ferdige, gjerne vinnende anbud, noe som er vanskelig å få tak i da bedrifter ikke er villig til å dele slik data. I tillegg trengs betydelige ressurser i form av prosesseringskraft, da store språkmodeller er svært ressurs-intensive å trene. Til slutt må en slik modell hostes på egen infrastruktur, eller på leid infrastruktur.

4.1.3 Alternativ løsning - Ingen forhåndsbehandling

Et annet alternativ til tjenersiden, kan være å ikke forhåndsbehandle data før vi sender det til modellen. Det kan gjøres ganske enkelt ved å legge til hele filer i meldingen til språkmodellen, som så leses av modellen. Dette kan være en enklere løsning, men ikke nødvendigvis bedre. Store språkmodeller sliter ofte med å finne riktig informasjon når de får inn mye data der ikke alt er relevant.

Ikke alle store språkmodeller støtter filopplasting, så ut av de diskuterte språkmodellene vil denne løsningen foreløpig kun være gjennomførbar med OpenAI sitt Assistants API. Dette API-et er i offentlig beta mens dette prosjektet utvikles, som vil si at det kan komme store endringer underveis i prosjektet eller etter at Link Utvikling har overtatt prosjektet.

4.1.4 Alternativ løsning - Lokal programvare

Et alternativ til å lage en nettportal er å lage en applikasjon som kunden installerer og kjører lokalt. En fordel med lokal programvare er at man slipper å hoste applikasjonen på nett, hvilket er kostnadsbesparende. Ulempen med å lage et slikt program er at man må ta hensyn til ulike operativsystemer under utviklingen, og at man må holde programvaren oppdatert slik at den blir kompatibel med nyere versjoner av alle operativsystemer.

I kombinasjon med alternativ løsning om å trene en egen modell, kunne kunden ha kjørt applikasjonen uten internett og lagret alle filer lokalt, hvilket er bra for personvern da man ikke trenger å lagre sensitive data i skyen, eller stole på tredjeparter. I tillegg vil en lokal modell være raskere da man ikke er avhengig av responstid fra API-er. Dette vil være en mye billigere løsning, da man ikke krever hosting, eller et dyrt API. Ulempen med dette er at det krever mye prosesseringskraft å kjøre en stor språkmodell lokalt, i tillegg til at selve modellen krever mye lagringsplass.

4.1.5 Diskusjon av alternativene

Dersom produktet skal utvikles med en egen språkmodell vil dette kreve mye tid, store mengder data og en høy oppstartkostnad, men føre til lavere kostnad i lengden. Grunnet tidsfrister og tilgjengelige ressurser er ikke dette alternativet aktuelt. Ved å ta i bruk en forhåndstrent modell, kan modellen benyttes med en gang uten å mate den noe form for treningsdata. Et mulig kompromiss er å bruke en forhåndstrent modell, og ta i bruk finjustering for å spisse modellen mot problemstillingen.

OpenAI og Anthropic har begge flere modeller som er tilgjengelige via deres respektive API-er. Begge leverandørene tar betalt for bruk av API-et, og ingen av leverandørene bruker brukerdata i trening av modellene når de benyttes gjennom API-et. Den mest vesentlige forskjellen mellom leverandørene er begrensning på antall kall man kan gjøre, der Anthropic har en mye strengere begrensning på mengde tokener man kan sende og antall kall som kan gjøres. Videre har vi Meta AI sine Llama modeller, disse er åpen kildekode og kan fritt brukes uten kostnad. Den eneste kostnaden ved bruk av Llama-modellene vil være å betale for hosting av modellen på en tjener, slik at man kan kommunisere med den over nett. Den beste Llama-modellen krever betydelige ressurser å kjøre og har et veldig lite kontekstvindu, og blir derfor ikke aktuell å bruke.

Ved å bruke OpenAI er det mulig å ta i bruk Assistants API for å laste opp hele filer uten forhåndsbehandling. Etter en del testing er det klart at dette er mulig, men modellen sliter med innlesing og forståelse av store filer. Det er mulig å forbedre ytelsen noe ved å spesifisere hvilke deler av filen modellen skal lese, men ettersom denne spesifiseringen er en prompt og ikke en faktisk begrensning på inndataene som sendes til modellen, er det uklart hvilken del av filen modellen velger å lese. Alternativ løsning om å ikke forhåndsbehandle filer forkastes derfor.

Ved forhåndsbehandling av filer lastes filen inn på tjenersiden, og omgjøres til ren tekst. Denne teksten kan renses og deles opp i mindre biter, såkalt *chunking*. Oppdelingen kan gjøres på mange måter, for eksempel semantisk eller ved hjelp av skilletegn som linjeskift og punktum. Videre kan disse bitene mates inn i en embedding-modell for å omgjøre dem til vektorer, hvilket åpner opp muligheter for vektorsøk. Denne metoden sørger for at mer relevante deler av teksten blir sendt med til modellen.

Det hadde vært mulig å lage en applikasjon som kjører lokalt på datamaskinen i stedet for en nettportal, noe som hadde vært bra for personvern, spesielt sammen med en egen modell som kjørte lokalt på maskinen. Siden det ikke er aktuelt å trene egen modell mister man en del av fordelene med lokal programvare. Applikasjonen må fortsatt bruke internett for å kommunisere med modellen, og vil fortsatt kreve brukerhåndtering for riktige tilganger og database. En lokal applikasjon bør også være multiplatform, og som et minimum støtte Windows og Mac-plattformene. En nettportal derimot vil virke på alle plattformer, uavhengig av hvilket operativsystem man bruker. Gruppen og oppdragsgiveren har erfaring med å lage webapplikasjoner, som gjør at det er det mest naturlige i dette tilfellet, spesielt siden det er minimal fordel med en lokal applikasjon ved bruk av kommersielt tilgjengelige modeller.

Når det kommer til valg av database, sto det mellom vektordatabase, dokumentdatabase, relasjonsdatabase eller en blanding av flere databasestrukturer. Vektordatabaser er spesielt godt egnet til prosjekter som har behov for semantisk søk, og vil egne seg godt til å blant annet finne ansatte med de nærmeste kvalifikasjonene til et prosjekt. Disse databasene er svært raske å utføre søk i, men er ofte kostbare. Dermed kan det være fornuftig å lagre kun data man trenger å gjøre semantisk søk på i en slik database, og benytte en annen type database for andre data. Dokumentdatabaser er fleksible og lar oss lagre informasjon som ikke passer inn i en vektordatabase, som for eksempel genererte utkast til anbud. Relasjonsdatabaser som Postgres er best egnet til data med en godt definert struktur, som ikke endres mye over tid. Siden utdata fra språkmodellen kan variere fra anbud til anbud, kan det være fornuftig å bruke en mer fleksibel database.

For vektordatabaser kan Pinecone være aktuelt, Pinecone er eksklusivt en vektordatabase. Det er mulig å få vektorsøk-funksjonalitet i flere databaser som ikke er utelukkende vektordatabaser. Et par eksempler på dette er utvidelsen *pgvector* til Postgres, eller MongoDB Atlas som støtter semantisk søk offisielt. Når det gjelder dokumentdatabaser så er det to aktuelle, Firebase Cloud Firestore og MongoDB. Cloud Firestore er kjent for gruppen og er lett å komme i gang med, mens MongoDB støtter vektorsøk i tillegg til lagring av vanlige dokumenter.

Siden prosjektet håndterer store mengder filer i forskjellige formater, kan det være aktuelt å laste disse opp til fillagring direkte uten å forhåndsbehandle dem for lagring i dokumentdatabase. Det kan derfor være aktuelt å bruke en form for *skylagring* for filer.

4.2 Valgt løsning

Etter en nøye vurdering av de forskjellige alternativene for løsningen, endte gruppen opp med en løsning basert på det initielle løsningsforslaget. Løsningen skal være en nettportal utviklet i Vue.js og Python, som benytter forhåndstrente språkmodeller fra OpenAI, med en kombinert vektor- og dokumentdatabase fra MongoDB Atlas, en skytjeneste til fillagring via Firebase Cloud Firestore og autentisering med Firebase Authentication. Modellen skal tilpasses ved å bruke prompter, og testes med ulike data for å sikre et godt resultat.

4.3 Valg av verktøy

I denne seksjonen blir valgt av verktøy beskrevet. Dette innebærer programvare, biblioteker, rammeverk, og store språkmodeller.

4.3.1 Programmeringsspråk, rammeverk og biblioteker

For tjenersiden av applikasjonen er programmeringsspråket Python tatt i bruk. Python ble valgt da det er et språk med god støtte for databehandling, og blir mye brukt i maskinlærings-prosjekter. For å lage endepunkter blir rammeverket Flask tatt i bruk, dette er et veldig enkelt rammeverk for å hurtig jobbe med webutvikling. For å samhandle med de store språkmodellene og vektordatabasen brukes rammeverket *LangChain*. LangChain har gode integrasjoner med alle verktøy som benyttes i prosjektet. Videre er Firebase Admin brukt for å samhandle med

Firebase Authentication og Firebase Cloud Storage. *Enhetstesting* utføres med Unittest biblioteket.

For å utvikle klienten benyttes det komponent-baserte rammeverket Vue.js, bygget på HTML, CSS og Javascript. Det er gunstig å benytte et rammeverk over ren Javascript da dette gjør utviklingsprosessen raskere og enklere. Vue.js er i tillegg godt etablert i markedet og har god støtte, god dokumentasjon og mange biblioteker tilgjengelig, blant annet PrimeVue som inneholder ferdige komponenter og ikoner. Rammeverket var også et ønske fra prosjekteier fordi Vue.js er noe de bruker i sine prosjekter, hvilket gjør at de kan bistå med sin ekspertise. TypeScript blir brukt for å ha sterke typer, noe som tillater enklere feilsøking, samt mer robust og lettleselig kode. TailwindCSS benyttes for å forenkle design av elementer i brukergrensesnittet. Enhetstester utføres med Vitest.

4.3.2 Databaser

For å utføre vektorsøk i dataene, benyttes en kombinert vektor- og dokumentdatabase fra MongoDB. Dette er en fleksibel løsning som gjør at man kan lagre embedding som et felt sammen med dokumentet embeddingen tilhører. Som et resultat av dette kan man filtrere vekk irrelevante dokumenter før man utfører vektorsøk, noe som ikke er mulig i en ren vektordatabase. Det er også mulig å lagre dokumenter som i en vanlig dokumentdatabase, uten embedding.

Før man kan gjøre vektorsøk i MongoDB, må man først definere en vektorsøk-indeks. En indeks i databasesammenheng, er en metode for å raskere søke i tabeller. Denne indeksen definerer hvilket felt i dokumentet som inneholder embeddingen, hvor mange dimensjoner embeddingen har, hvilken metrikk som skal brukes, og hvilke felter man kan filtrere på. Det er viktig at dimensjonen som blir definert i indeks, er den samme som man får fra valgt embeddingmodell. Når man skal utføre et vektorsøk, velger man indeksen som er definert, og eventuelle forhåndsfiltre for å filtrere søket.

Firebase Cloud Storage benyttes for å lagre opplastede og genererte filer. Dette er en skylagringstjeneste som er integrert med andre tjenester og API-er fra Firebase, blant annet Firebase Authentication, noe som forenkler styringen av tilgang til filer. Denne skylagringstjenesten inneholder også versjonering av filer, noe som kan være nyttig dersom kunden laster opp en nyere versjon av et dokument fra en anbudskonkurranse.

4.3.3 Store språkmodeller

Som språkmodell er OpenAI sine GPT-modeller valgt grunnet tilgjengelighet, kvalitet på API, størrelse på kontekst-vindu, pris og ytelse. Modellene GPT-3.5 Turbo og GPT-4 Turbo vil brukes til uthenting og generering av tekst. GPT-3.5 Turbo modellen er mye billigere og raskere enn GPT-4 Turbo, derfor ønsker man å bruke denne mest mulig. Dersom det skal genereres ny tekst, gir GPT-4 Turbo også vanligvis bedre resultater, så denne vil brukes i tilfeller der GPT-3.5 Turbo ikke gir tilstrekkelig resultat (OpenAI, 2024e). Embedding-modellen Text Embedding 3 Small brukes til å lage embeddinger. Denne modellen har en lav pris i forhold til de andre tilgjengelige modellene, og skal gi et nesten like godt resultat (OpenAI, 2024b).

4.3.4 Utviklingsmiljø

Det er brukt forskjellige utviklingsmiljø til forskjellige deler av prosjektet.

På klientsiden falt valget på Visual Studio Code (VS Code), et program for redigering av kode med god støtte for forskjellige programmeringsspråk. VS Code har god støtte for rammeverket Vue.js, som brukes til utvikling av applikasjonens klient.

På tjenersiden blir PyCharm brukt. PyCharm er et integrert utviklingsmiljø til Python, som gir kodeanalyse, en grafisk debugger, en integrert enhetstester og integrasjon med versjonskontrollsystemer.

For eksperimentering og utforsking av data har gruppen tatt i bruk Google Colab, en nettbasert kode-editor for Jupyter Notebooks. Jupyter Notebooks blir brukt for å lage interaktive "notatbøker" som inneholder kode, formler, visualiseringer, tekst og viser resultatet av kjøringen etter hver kodeblokk. Slike notatbøker er svært fleksible og egner seg godt til eksperimentering.

Som versjonskontrollsystem er GitHub tatt i bruk. GitHub lar gruppen lagre, flette sammen, versjonere og samarbeide om kode.

4.3.5 Samarbeidsverktøy

Jira er et verktøy for smidig prosjekthåndtering som støtter alle agile utviklingsmetoder. Verktøyet inneholder tavler, køer, tidslinjer, saker og timeføring. Dette verktøyet tillater gruppen å holde oversikt over hvilke oppgaver hvert gruppemedlem holder på med, og hva som må gjøres når.

Til brainstorming, huskelapper, utkast av diagrammer, lenking til diverse dokumenter og lignende har verktøyet Miro blitt brukt. Miro er en digital samarbeids-tavle der et team kan tegne, skrive og legge inn Post-it lapper i sanntid.

For daglig kommunikasjon brukte gruppen Discord, en chat-applikasjon med tjenere og gruppesamtaler. Link Utvikling har en tjener for kommunikasjon innad i bedriften, og prosjektgruppen har en egen gruppesamtale for å avtale møtetider og annen kommunikasjon.

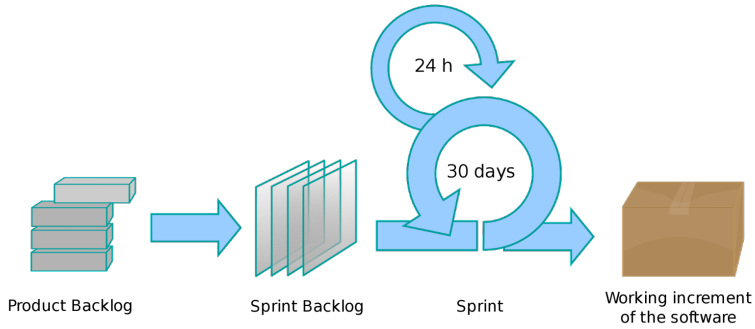
4.4 Prosjektmetodikk

4.4.1 Utviklingsmetodikk

Prosjektet involverer utvikling av komplekse systemer som benytter ny teknologi i stadig utvikling. I tillegg er ikke alle krav avdekket ved prosjektstart og krav kan endre seg over tid. Det er dermed hensiktsmessig med en smidig og iterativ metode for utvikling som sikrer fleksibilitet og tilpasningsevne etter hvert som krav og behov avdekkes eller endres. Dette kapittelet gjennomgår aktuelle utviklingsmetoder og diskuterer valgt metode.

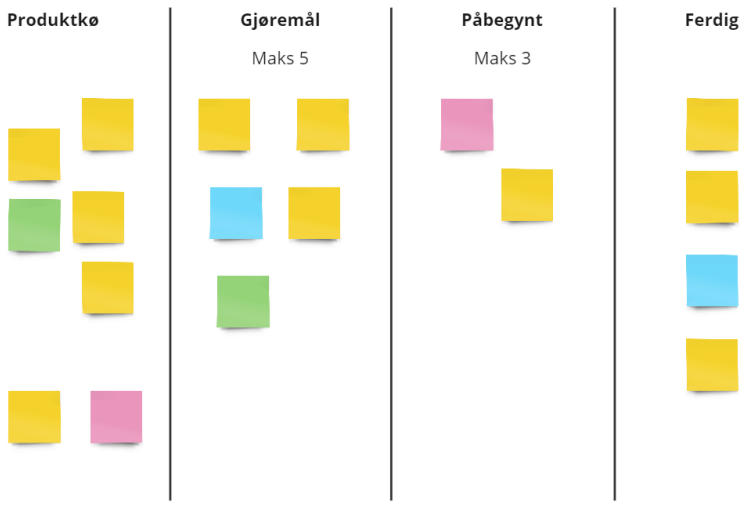
Scrum er et agilt rammeverk som gir en tilpasningsdyktig og inkrementell tilnærming til utvikling av programvare (Schwaber og Sutherland, 2020). Metodikken inneholder en produktkø, der arbeidet som skal utføres er brutt ned i små oppgaver, ofte i form av brukerhistorier og brukstilfeller. Disse omfatter funksjoner, feilrettinger og andre krav for

produktet. Arbeidet gjennomføres i tidsbokser kalt *sprinter*, der hver sprint har en varighet på mellom en uke og en måned. En sprint starter med sprintplanlegging, der målet for sprinten blir avklart og oppgaver hentes fra produktkøen. Hver sprint avsluttes med to møter, en sprintgjennomgang og et sprintretrospektiv. Under sprintgjennomgangen vises fremdriften til interessenter for å få tilbakemelding, og under sprintretrospektiv diskuterer man lærdom og forbedringer til neste sprint. Scrum oppfordrer også til tett samarbeid ved daglige møter, kalt daglig scrum, der teamet oppdaterer hverandre om fremdrift og problemer (Figur 4-1).



Figur 4-1 - Scrum-prosessen (Lakeworks, 2024).

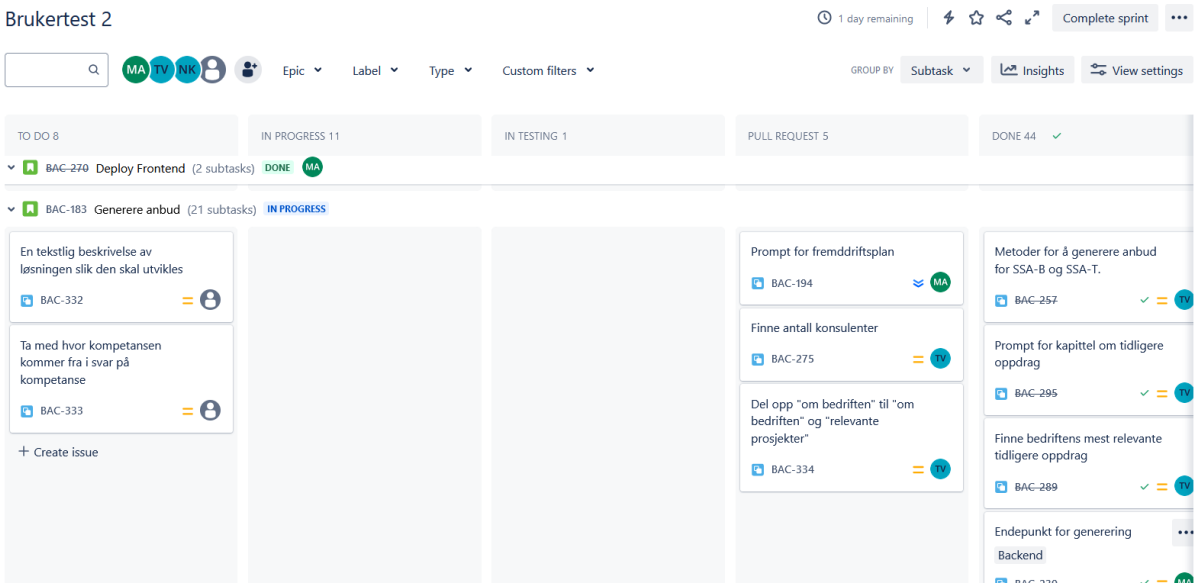
Kanban er et agilt rammeverk som visuelt representerer arbeid som skal utføres på et kanban-brett (Rehkopf, 2024). Kanban-brett inneholder fem komponenter: kort, kolonner, maksgrense for kolonner, forpliktelsespunkt og leveransepunkt. Kortene inneholder en arbeidsoppgave hver, og starter i en produktkø. Brettet inneholder kolonner, der hver kolonne representerer et steg i arbeidsflyten. En slik arbeidsflyt kan være så enkel som "Gjøremål", "Påbegynt" og "Ferdig", men settes opp ut fra prosjektets behov (Figur 4-2). Det blir satt maksgrenser for kolonner, slik at igangsatt arbeid må gjøres ferdig før nytt arbeid kan påbegynnes. Kort kommer inn på brettet fra forpliktelsespunktet ("Gjøremål" i eksempelflyten), går gjennom alle kolonnene, og ender opp på leveransepunktet ("Ferdig" i eksempelflyten) når oppgaven er ferdig.



Figur 4-2 - Kanban-brett.

For å legge til rette for god kommunikasjon og fleksibilitet, samt visualisere oppgaver og arbeidsmengde, har en kombinasjon av Kanban og Scrum blitt brukt for prosjektet. Arbeidet har blitt utført i sprinter på to uker, der tydelige mål for iterasjonen blir satt under sprintplanlegging. Teamet har en Scrum master som leder retrospektiv og planleggingsmøter, men ellers ingen formelle roller. Under planlegging hentes arbeidsoppgaver fra produktkøen, som kontinuerlig oppdateres og forbedres av alle teammedlemmer. Under en sprint kan oppgaver omprioriteres, omfordeles og oppdateres etter behov. Ved slutten av en sprint gjennomføres et retrospektiv-møte der utført arbeid gjennomgås, lærdom og forbedringer til neste sprint diskuteres, produktkøen raffineres, og neste sprint blir planlagt. Eventuelle uferdige oppgaver eller oppgaver som går på tvers av flere sprinter, videreføres til neste sprint. Det blir ikke gjennomført daglige møter, i stedet blir oppgavetavlen brukt for å visualisere hvilke oppgaver som er under arbeid (Figur 4-3), og kontinuerlig kommunikasjon gjennom arbeidsdagen blir brukt for å løse problemer med oppgaver. Teamet har jevnlige møter med intern og ekstern veileder for tilbakemelding, diskusjon og avklaring av spørsmål.

Denne kombinerte utviklingsmetoden er tilpasset behovene for prosjektet og teamet. Sprinter fra Scrum fører til klare mål og prioriteringer i hver iterasjon, mens den mer fleksible oppgavehåndteringen fra Kanban lar teamet raskt omstille seg for å tilpasse utviklingen til nye krav eller ny lærdom. Ved å bruke oppgavetavlen som en visuell oversikt over hvem som jobber med hva, kan daglige møter droppes.



Figur 4-3 - Oppgavetavle i tjenesten Jira.

4.4.2 Prosjektplan

Prosjektplanen (Vedlegg 3, Figur 1.1) fremlegger en overordnet plan for prosjektet i form av et GANTT-diagram, med milepæler, oppgaver og oversikt over sprinter som skal gjennomføres. For hver oppgave er det planlagt varighet, både i uker og i arbeidstimer. Oppgavene er delt inn i to distinkte kategorier, "Internt" og "Utvikling", hvor oppgaver under "Internt" omfatter dokumentasjon, presentasjoner og andre arbeidskrav. Oppgaver under "Utvikling" omfatter utforsking, prompt engineering, utvikling, brukertesting og lansering.

Det er totalt planlagt 11 sprinter på to uker for prosjektet, der hver sprint omfatter oppgavene som løper i de tilhørende ukene.

Prosjektet har 9 milepæler som markerer viktige hendelser i prosjektets løp, som blant annet ferdigstilling av prototype, brukertesting, innlevering av rapport og lansering av løsning.

4.4.3 Risikovurdering

I risikovurderingen (Vedlegg 3, Tabell 2-1) har gruppen satt opp en vurdering av ulike risikomomenter som kan inntreffe under prosjektets løp. Herunder tekniske risikoer og risikoer innad i gruppen.

En av risikoene som er blitt vurdert som høyest er risikoen for at OpenAI sitt API får endringer underveis som fører til at applikasjonen må endres. OpenAI API-et har vært på markedet siden 2022 (OpenAI, 2022), men er fremdeles i stadig utvikling. GPT-4 Turbo, en av modellene som er tatt i bruk i prosjektet, er i offentlig beta ved starten av prosjektet og kan komme med endringer underveis. Denne modellen er i slutten av testfasen og vil komme til å bli lansert som en stabil modell innen slutten av prosjektet. Dette vil mest sannsynlig ikke føre til store endringer i selve API-et, men kan føre til endringer i utdata fra modellen.

Videre har gruppen vurdert faren for å estimere feil tid på de oppgavene som skal gjøres. Dette er vurdert som sannsynlig fordi ingen av medlemmene har mye erfaring med prosjektplanlegging fra før, noe som tilsier at tidsestimatene gruppen har gjort, mest sannsynlig vil være feil. Tiltaket gruppen har gjort er å være generøs med tid på de oppgavene som vil være mest utfordrende.

Kompleksiteten til store språkmodeller kan også være en risiko som kan føre til at løsningen må endres, enten fullstendig eller delvis. Det kan være tidkrevende og komplisert å utforme gode prompter, særlig med tanke på at man ikke med sikkerhet kan vite hvordan de forskjellige delene av prompten påvirker utdata fra språkmodellen. Det er derfor satt av god tid på prompt engineering, så gruppen får tid til å utforme effektive prompter.

Mangelen på testdata er også en vesentlig risiko. Det er behov for å kunne se eksempler på anbudskonkurranser og svar på disse. Anbudskonkurransene i seg selv er ofte offentlig tilgjengelig, men svarene på konkurransene inneholder i mange tilfeller bedriftshemmeligheter, og er unntatt offentlig tilgjengelighet. I dette tilfellet må gruppen selv ta kontakt med andre bedrifter om å få se anbudene som de har sendt ut, eller be om innsyn i anbudskonkurranser i kommunen. Dermed er mangelen vurdert som sannsynlig, da de er unntatt offentlig tilgjengelighet. Konsekvensene av denne mangelen er problemer med testing og evaluering, da man har mangel på data å sammenligne med.

4.5 Evalueringsplan

For å vurdere design og omfang av produktet vil kontinuerlig tilbakemelding gis av oppdragsgiver Link Utvikling ved møter annenhver uke. Link Utvikling stiller med to utviklere som bidrar med tilbakemelding og testing underveis. I tillegg vil det gjennomføres brukertester med eksterne og interne testere, og øvrig kode har enhetstester.

4.5.1 Brukertesting

Prosjektet er preget av funksjonalitet som er vanskelig å teste på en objektiv måte. En viktig del av evalueringen vil dermed være de subjektive opplevelsene til brukere. For å kunne si noe om hvor god løsningen er, og om produktet er forbedret siden forrige brukertest, må man utarbeide spørsmål relatert til kvalitet og brukbarhet, som kan brukes i testene.

Den første strukturerte brukertesten skal gjennomføres når hovedfunksjonaliteten til systemet er på plass, innen uke 14. Denne brukertesten utføres internt, med utviklere fra Link Utvikling som testere, med fokus på systemet i sin helhet og prosessen med å generere anbud.

Etter brukertesten skal produktet forbedres, og en ny brukertest gjennomføres to uker senere, i uke 16. Denne brukertesten skal kun omhandle kvalitet på generert tekst.

En av de endelige brukertestene utføres i uke 18 med eksterne testere fra bedriften FunBit, som Link Utvikling har inngått en avtale med. I tillegg vil en brukertest gjennomføres med konsulentselskapet Rainfall, som gruppen selv har avtalt test med. Denne testen skal gjennomføres i løpet av uke 19. I disse brukertestene vil hele produktet vurderes, både kvalitet på generert utdata og brukervennligheten til systemet.

Målet med disse brukertestene er å se hvor brukervennlig systemet er, og hvor høy kvalitet det er på den genererte teksten i forhold til forventningene fra brukerne. På denne måten er det mulig å sammenligne kvaliteten på applikasjonen fra test til test, og det vil til slutt være mulig å si noe om hvor fokus burde ligge videre i utvikling av applikasjonen.

4.5.2 Enhetstester

Kodebasen skal testes underveis ved hjelp av enhetstester, til både tjenerside og klientside. Dette gjelder først og fremst logikken i tjenersiden, men kan også inkludere noe testing av grafisk grensesnitt og logikk i klientsiden. Testing mot database og andre API-er krever *mocking* av data, for å sikre at testene alltid gir samme resultat og ikke endrer data som eksisterer i databasen eller påfører høye kostnader ved bruk av API. Dette er for å få tilbakemelding med en gang på om enheter virker som de skal eller ikke.

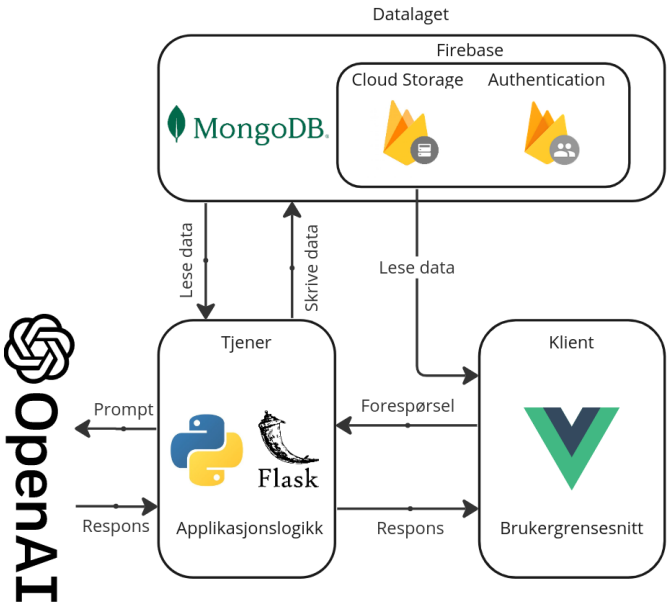
Gruppen har satt et mål på minst 80% testdekning for tjenersiden av applikasjonen. Ettersom klientsiden først og fremst er brukergrensesnitt og inneholder lite logikk, har ikke gruppen satt et testdekningsmål der.

4.5.3 Likhetstesting

Siden utdata fra store språkmodeller er uforutsigbare i og med at du kan få forskjellig svar med samme inndata, er det umulig å lage enhetstester som tester hvor bra utdataene er. Dette må dermed gjøres delvis manuelt, ved å få mennesker til å gå gjennom resultatene. Der det er hensiktsmessig vil det også utføres *likhetstesting*, der man sammenligner to tekster for å se hvor like de er.

5 DETALJERT LØSNING

Nettportalens arkitektur baserer seg på klient-tjener-arkitekturen. Dette er en godt kjent arkitektur for distribuerte applikasjoner som fordeler oppgaver mellom klienter og tjenere, der klienter sender forespørsler om tjenester fra tjenere, og tjenerne leverer tjenester til klienter. I tillegg til klient og tjener-lagene er det to ekstra lag i arkitekturen. Det ene er et datatilgangslag som består av MongoDB-databasen, Firebase Cloud Storage og Firebase Authentication. Det andre er OpenAI sitt API som tjener kommuniserer med. En oversikt over systemarkitekturen, som illustrerer hvordan de ulike delene kommuniserer med hverandre, er avbildet i Figur 5-1. Tjeneren er den mest sentrale delen av applikasjonen, og vil styre flyten til systemet mellom de ulike delene, med unntak av Firebase, som klienten kan sende forespørsler til direkte.



Figur 5-1 - Systemarkitektur.

5.1 Database og filhåndtering

5.1.1 Skylagring (Blob Storage)

For å lagre hele filer slik at de kan hentes når det er behov for det, brukes Firebase Cloud Storage. Cloud Storage er en ustrukturert skylagringstjeneste for binære filer, som er nyttig for å lagre anbudskonkurransefiler, slik at de kan hentes ut på et senere tidspunkt. Filene er plassert inn i mapper, hvor den første mappen bruker ID-en til bedriften som eier filene. De etterfølgende mappene bruker ID-en til anbudskonkurransen filene tilhører. Dette er for å sikre at det ikke blir noen navnekonflikter mellom ulike anbudskonkurranser, og at det blir lett å hente dem ut igjen.

5.1.2 Oppdeling av data i biter

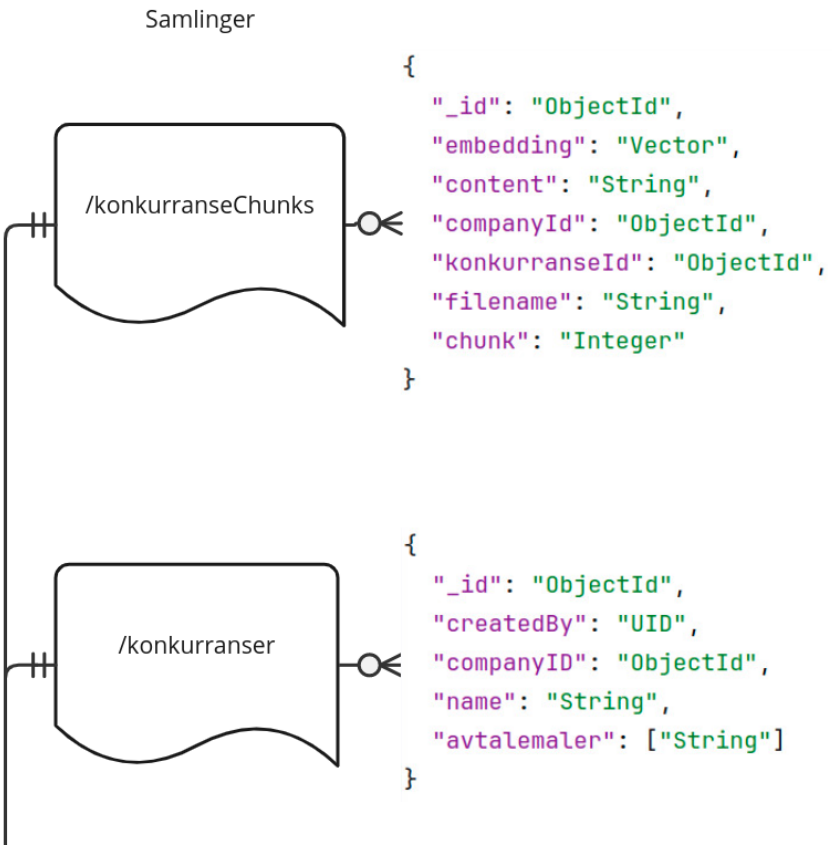
Når anbudskonkurransefiler blir lastet opp til tjeneren, hentes innholdet fra dokumentet ut til ren tekst som språkmodeller kan jobbe med, ved hjelp av UnstructuredDocumentReader og pypdf, biblioteker som leser filer til ren tekst som språkmodellene kan jobbe med. Etter innholdet er hentet ut, blir det delt opp i biter hvor hver bit blir embeddet. Dette er for å gjøre det lettere å søke på store dokumenter, finne relevante biter, og det sikrer at dokumentet får plass i modellens kontekstvindu. Filene blir delt opp i biter på en lengde av 2000 tokener, hvor 100 av dem overlapper med forrige bit. Denne overlappingen er nødvendig siden algoritmen kan avslutte en bit midt i en setning eller et sammenhengende tema, som fører til tap av kontekst. Ved å ha denne overlappingen vil en av bitene inneholde hele setningen. Disse tallene har blitt valgt etter testing og med utgangspunkt i et den totale bruken av tokener ikke vil føre til at utdata overstiger 4096 tokener. Dersom man for eksempel sender 16 000 tokener i inndata, der man har behov for omtrent 5000 tokener i utdata, vil ikke modellen klare å generere et fullverdig svar. Da er det gunstig å dele opp inndata i to bolker på 8000 tokener, med en forventet respons på 2500 tokener. Man må altså ta hensyn til hvor mye utdata man har behov for i forhold til lengden på inndata.

5.1.3 Strukturering av databasen

MongoDB Atlas-databasen er delt opp i totalt fem ulike samlinger, som lagrer data til ulike formål: konkurranser, konkurransebiter, konsulenter, bedrifter og anbud.

Konkurransesamlingen inneholder all konkurransedata som bedrifter har lagt inn. For å skille mellom hvilke data som tilhører hvilken bedrift, har hvert dokument en ID til bedriften som har laget den, noe som gjør at man kan filtrere på konkurransene til en bedrift. En tilhørende samling er konkurransebiter-samlingen, som inneholder *dokumentbiter* fra konkurranse-dokumenter som er lastet opp. Hver bit inneholder en egen embedding for å kunne søke i samlingen med vektorsøk. Figur 5-2 viser et utsnitt av databasen og hvordan disse feltene er definert.

I tillegg til konkurransedata må det lagres data om konsulenter, bedrifter og anbud som er generert. Konsulentdataene er stort sett data man finner i en CV, og inneholder blant annet arbeidserfaring og tidligere prosjekt. For å kunne utføre vektorsøk på konsulenter, så er det også lagret en embedding av disse dataene. Bedriftsdata inneholder litt generell informasjon om en bedrift, som hvilke tjenester de tilbyr og tidligere prosjekter de har jobbet med. Til slutt har man anbudsamlingen som inneholder genererte anbud for en konkurranse, der hver konkurranse kan ha ett anbud lagret.



Figur 5-2 - Utsnitt av database, her konkurranse- og konkurranseChunks-samlingene.

5.1.4 Søk i database

Konsulentdata og biter fra anbudskonkurransefiler har et felt som inneholder embedding til dokumentet. Dette feltet brukes når man utfører et vektorsøk i samlingen. Vektorsøk gir oss de *k* nærmeste treffene sammen med en poengsum for hvert treff, som forklarer hvor stor semantisk likhet det er mellom spørringen og hvert resultat. Høyere tall tilsier høyere relevans. Det kan i noen tilfeller være fornuftig å hente flere enn ett dokument, ettersom informasjonen man trenger kan være delt opp over flere dokumenter.

Det er noen tilfeller hvor man ikke ønsker å søke på alle dokumenter, bare de som oppfyller visse krav. I MongoDB Atlas kan man bruke *forhåndsfiltrering*, hvor man kan spesifisere felter man ønsker å filtrere på før man søker. Altså, kun de dokumentene som oppfyller kravet til filteret, blir utført vektorsøk på. Dette er nødvendig da alle bedrifter har sine dokumenter i en felles samling, med ID-en til bedriften for å skille mellom de ulike bedriftene sine dokumenter. Dette gjør det også mulig å filtrere på andre felter som filnavn, for å kun utføre søk på dokumenter som tilhører samme opplastede fil.

Konsulenter i en bedrift er også lagret i databasen med en egen embedding som gjør vektorsøk mulig. Man kan for eksempel søke etter et programmeringsspråk, og få tilbake de mest relevante ansatte for denne kompetansen, selv om kompetansen ikke er nøyaktig det som søkes etter. For eksempel vil man kunne få tilbake en ansatt med erfaring i JavaScript dersom kompetansen som etterspørres er React. Søk i ansatte skal kun gjelde ansatte for den aktuelle

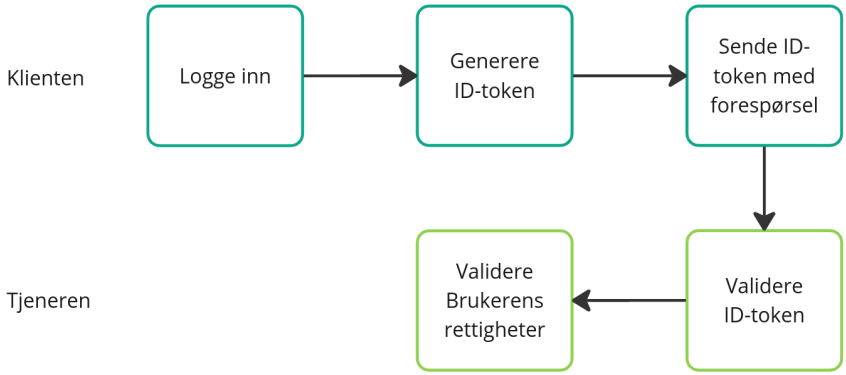
bedriften, og det blir derfor utført forhåndsfiltrering på bedrift for å fjerne ansatte fra andre bedrifter fra søket.

5.2 Brukerhåndtering

5.2.1 Autentisering og autorisering

For autentisering brukes Firebase Authentication, en ferdiglaget løsning som inneholder lagring av brukere, hashing og salting av passord og en nettportal i Firebase Console, hvor man kan administrere brukere. Det kommer også med API til både tjener- og klientsiden for å lage / endre brukere, logge inn og andre relevante funksjoner. Dette inneholder også metoder for å sette opp autorisering, hvor man kan definere ulike tilganger til ulike ressurser.

Figur 5-3 viser i grove trekk hvordan autentisering fungerer. For at brukeren skal få tilgang til ressursen må hele prosessen være fullført. Brukeren starter ved å logge inn med sin egen e-postadresse og passord. Da vil brukeren få tilgang til sider som den skal ha tilgang til, helt til man logger ut eller sletter nettleserdata. De sidene som brukeren ikke har tilgang til, vil gjøre en omdirigering tilbake til startsiden. For hver forespørsel som blir utført etter at brukeren er logget inn, så vil det genereres en ID-token dersom det ikke finnes en allerede, som er gyldig i kun én time. Denne nøkkelen blir sendt sammen med forespørselen til tjeneren, som vil validere at denne nøkkelen faktisk eksisterer og fremdeles er gyldig (Firebase, 2024b). Dersom den er gyldig får man tilgang til brukerdataene til brukeren, da kan man utføre autorisering ved å sjekke både bedrifts-ID og rollen til brukeren for å sikre at kun de som har riktige roller og tilhører riktig bedrift, får tilgang til ressursen.



Figur 5-3 – Brukerautentisering og autorisering.

5.2.2 Brukerroller

Nettportalen bruker tre ulike roller: *bruker*, *bedriftadmin* og *systemadmin*. Hvilken rolle brukeren har og ID til bedriften brukeren tilhører, er lagret som "*custom claims" i* Firebase Authentication. Custom claims er ekstra felter som tilhører hver enkelt bruker og blir ofte brukt for autorisering (Firebase, 2024a), dette gjør at man slipper å lagre brukerdata i en egen database.

Alle brukere har kun tilgang til sin egen bedrift, og hvilken data de kan se er avhengig av brukerens rolle. Vanlige brukere har kun mulighet til å opprette nye konkurranser, samt

redigere eller slette konkurranser brukeren selv har laget. Dette gjøres ved å lagre ID til brukeren som opprettet konkurransen, sammen med konkurransen i databasen. De skal også ha mulighet til å oppdatere sin egen brukerprofil.

En bedriftadmin vil ha samme tilgang som en vanlig bruker, men vil også ha tilgang til konkurranser som andre brukere i bedriften har laget. Bedriftadmin har også tilgang til å håndtere alle konsulentene i bedriften, dette inkluderer å lage nye, enten ved å fylle inn feltene manuelt, eller ved å laste opp en ferdig utfylt CV, samt redigering og sletting av eksisterende konsulenter. Hver bedrift må ha minst en bedriftadmin som kan legge inn konsulenter. Bedriftadmin kan også opprette og slette brukere innad i egen bedrift, samt opprette nye bedriftsadministratorer.

En systemadmin har alle rettigheter en bedriftadmin har, men har i tillegg tilgang til å opprette nye bedrifter og vedlikeholde eksisterende. En systemadmin kan også lage bedriftadmin-brukere for andre bedrifter, og legge til flere. Systemadmin vil i utgangspunktet være eier av produktet, og det må være minst en bruker som har denne rollen.

5.2.3 Firebase Security Rules

Firebase har sitt eget sikkerhetslag som gjør at det er mulig å gjøre databasekall og lignende fra en klient, uten en dedikert tjener for det. Dette laget kalles *Firebase Security Rules* og kan definere ulike regler som returnerer sann eller usann. Kun dersom den relevante regelen returnerer sann, vil brukeren få tilgang til ressursen. Disse reglene blir kun brukt når man sender eller henter data fra klienten, ikke fra tjeneren. I dette prosjektet er det kun aktuelt å definere disse reglene for Firebase Cloud Storage, siden det er den eneste Firebase-tjenesten utenom Firebase Authentication som benyttes i prosjektet. Den første regelen er definert til å gi lesetilgang til en fil dersom brukeren er logget inn, tilhører samme bedrift som filen og filen tilhører en konkurranse som brukeren har opprettet (Figur 5-4). Dersom brukeren er en bedriftadmin, har man tilgang til alle filer i sin bedrift. Den andre regelen gjelder bare for å liste opp filer, alle ansatte i bedriften har tilgang til dette. Dette gir ikke tilgang til selve filene, bare filnavn og metadata.

```
match /b/{bucket}/o {
  match /{companyId}/{konkurranseId}/{filename} {
    allow get: if isSignedIn() && companyMatches(companyId) && createdByMatches();
  }
  match /{companyId}/{konkurranseId}/{allPaths=**} {
    allow list: if isSignedIn() && companyMatches(companyId);
  }
}
```

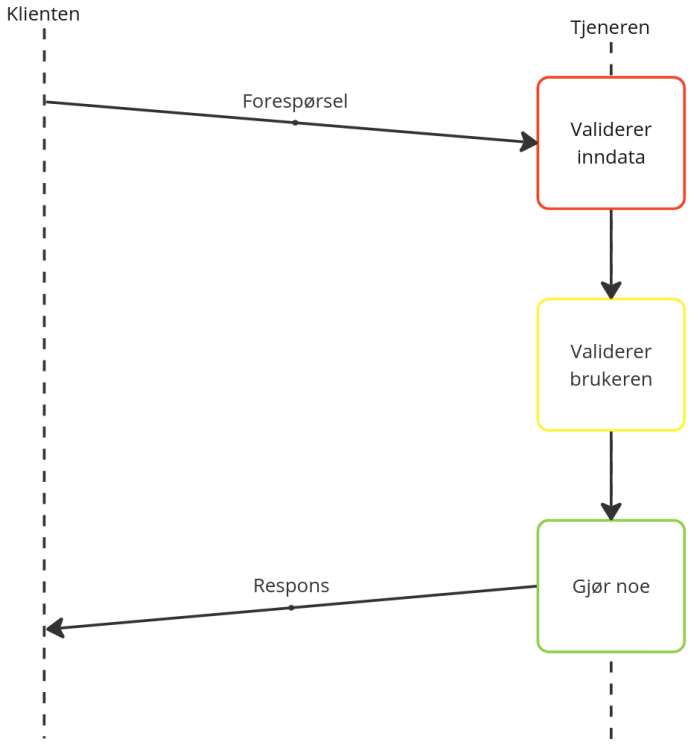
Figur 5-4 - Firebase Security Rules.

5.3 Tjeneren

5.3.1 API og Validering av data

For å sette opp endepunktene er rammeverket Flask brukt, sammen med Flask-OpenAPI3 biblioteket. Denne kombinasjonen gjør at vi enkelt kan sette opp endepunkter som støtter automatisk validering av inndata med biblioteket Pydantic, og som automatisk genererer OpenAPI-dokumentasjon. Pydantic gjør det enkelt å validere data ved å definere dem som en klasse. For eksempel kan man definere en modell som tilsvarer data som ligger i MongoDB og validere at alle endringer passer inn i den modellen. Dette er viktig å gjøre for å sikre at data blir lagret på riktig format. OpenAPI dokumentasjon er også nyttig for å dokumentere endepunktene som gjør det lettere å ta i bruk API-et.

Et typisk API-kall til tjeneren kan se ut som i Figur 5-5, hvor data først valideres med Pydantic etterfulgt av validering av brukeren. Til slutt utfører funksjonaliteten på endepunktet, for eksempel hente data fra databasen, før den sender en respons tilbake til klienten. Dersom validering eller noe annet feiler, vil responsen bli sendt tidlig med en feilmelding og en HTTP-statuskode. Tjeneren bruker REST-arkitekturen som vil si at hver forespørsel er uavhengig fra andre forespørsler, og man kan identifisere ulike ressurser med en unik ID.



Figur 5-5 - Forespørsler mot tjener.

5.3.2 Service

Tjeneren inneholder service-klasser for å utføre CRUD-operasjoner (Opprett, les, oppdater og slett), spørringer og vektorsøk mot databasen. Hver samling i databasen har en tilhørende service-klasse med CRUD-operasjoner som minimum. Samlinger med embedding har i tillegg metoder for vektorsøk.

5.3.3 Chains

Chains-klassene inneholder metoder for å sende API-kall til OpenAI API-et ved hjelp av LangChain Chains. LangChain er et rammeverk for utvikling av applikasjoner som bruker store språkmodeller. Rammeverket forenkler flere deler av utviklingen, ved å tilby forskjellige "byggeklosser" og komponenter som kan brukes uavhengig av språkmodell-leverandør eller modell. I dette prosjektet er to av hovedfunksjonalitetene til rammeverket tatt i bruk, Chains og PromptTemplates.

PromptTemplates er forhåndsdefinerte oppskrifter for å generere prompter (Figur 5-6). Slike maler gjør det mulig å sette variabler inn i en prompt, slik at man slipper å definere en ny prompt hver gang man har ny data å sette inn. I dette prosjektet er kun den mest grunnleggende PromptTemplate blitt brukt, som lar oss definere inndata-variabler, template (prompt med variabler, Figur 5-7) og partielle variabler, som er variabler som kommer fra resultatet av en funksjon.

Chains gjør det enkelt å utføre en rekke kall etter hverandre, enten det er til en språkmodell, forhåndsprosessering eller formatering. LangChain tilbyr LangChain Expression Language (LCEL), for å utføre dette. Med LCEL går det raskt å for eksempel sette opp kjeder som først legger dataene inn i en PromptTemplate, så sender prompten til språkmodellen, for så å til slutt formatere utdataene til et gitt format (Figur 5-8).

```
PromptTemplate(
    template=amount_of_consultants_template,
    input_variables=["context"],
    partial_variables={"format_instructions": parser.get_format_instructions()}
)
```

Figur 5-6 - PromptTemplate

```
amount_of_consultants_template = """
You are given a list of file contents from an tender competition.
Your task is to find out how many consultants that are needed to complete the job.
You should scan the given text for any mention of the amount of consultants needed.
If the amount is not explicitly mentioned in the text, you should leave the field empty.

The text to search in:
```{context}```

The format of the response should be:
```{format_instructions}```
"""
```

Figur 5-7 - Template for bruk i en PromptTemplate

```
chain = prompt | self.llm | parser
```

Figur 5-8 - LangChain Expression Language (LCEL)

5.3.4 Forretningslogikk

Forretningslogikk-klassene fungerer som et mellomledd mellom API-et, Service-klasser og Chains-klassene. Disse klassene samler all nødvendig informasjon fra API og database, og sender disse dataene til Chains-klassene for å generere svar med språkmodellene. Deretter samler forretningslogikk-klassene resultatene fra Chains-klassene og returnerer svaret til API-et.

5.3.5 Asynkrone metoder

Siden flere metoder i applikasjonen krever mange kall til OpenAI API-et, vil det ta veldig lang tid å sende dem en etter en. Ved å gjøre disse kallene asynkront kan man samle alle API-forespørsler som skal gjøres samtidig og sende de fortløpende. Dette gjør at tjeneren blir ledig til å utføre andre oppgaver mens de asynkrone kallene venter på respons, og tjeneren slipper å vente på respons fra hvert API-kall før neste sendes. De asynkrone kallene samles opp og resultatet leveres samlet når alle kall er ferdig.

5.3.6 Testing

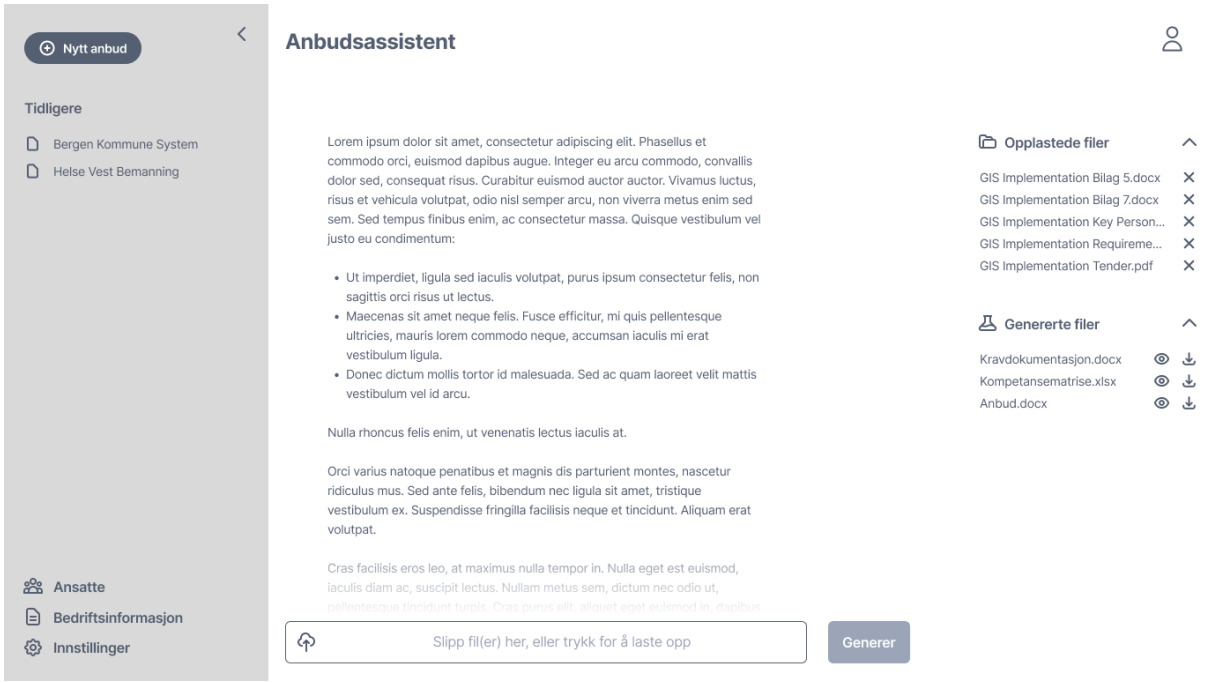
For testing i tjeneren brukes først og fremst enhetstester med unittest-rammeverket. Disse tester metoder i serviceklasser, endepunkter og andre funksjoner, for å sikre at de alltid gir riktig utdata.

For å unngå at testene påvirker API-ene som brukes i forhold til kostnad, responstid og lignende, er en del metoder "*mocket*", der man definerer resultatet for å unngå sideeffekter med API-er. For å mocke er det brukt ulike metoder, den ene er å bruke unittest sine innebygde funksjoner for mocking av data, hvor man selv definerer returverdi eller sideeffekter. Den andre metoden er et bibliotek kalt *Mongomock*, som brukes for å teste mot MongoDB, ved å lage en falsk MongoDB database som kjører lokalt og ikke permanent lagrer data. Dette sparer en del tid og gjør det lettere å unngå feil, da man slipper å definere returverdier på egenhånd.

5.4 Klienten

5.4.1 Brukergrensesnitt

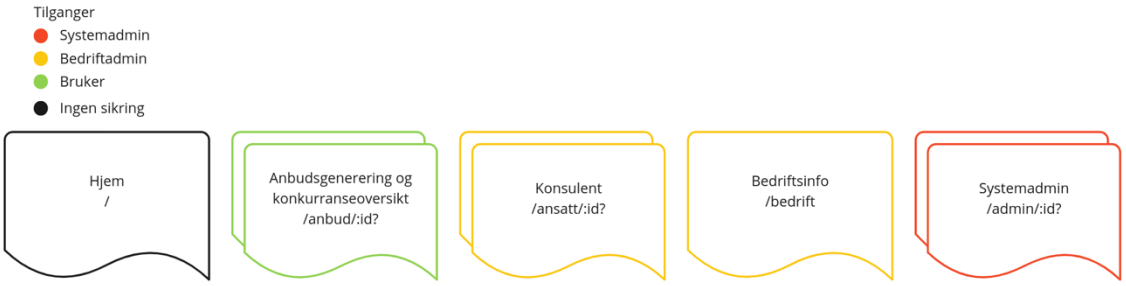
Brukergrensesnittet til nettportalen er enkel og gjenkjennelig for de som har brukt ChatGPT. Sidefeltet på venstre side er en oversiktsvisning, nederst på sidefeltet ligger navigasjonsknapper til de ulike rutene i applikasjonen. Det midterste feltet er reservert til hovedinnholdet på siden, blant annet tekst-utdata fra modellen. Høyre side er reservert for ekstra informasjon som tilhører valgte side. Brukergrensesnittet er satt opp for å sette mest mulig fokus på anbudsfunksjonaliteten i nettportalen, da dette er hovedfunksjonaliteten til løsningen. Figur 5-9 viser en tidlig skisse over hovedsiden til nettportalen, hvor man kan laste opp filer og generere anbud.



Figur 5-9 - Figma-skisse for anbudsgenerering.

5.4.2 Ruter

Klienten inneholder flere ruter med ulike tillatelser og funksjoner. Figur 5-10 viser en enkel oversikt over disse, samt hvilke brukerroller som har tilgang til dem. Rutene som er merket med ":id?", er dynamiske ruter hvor ulikt innhold kan vises basert på innholdet i URL-en.



Figur 5-10 - Ruteoversikt klient

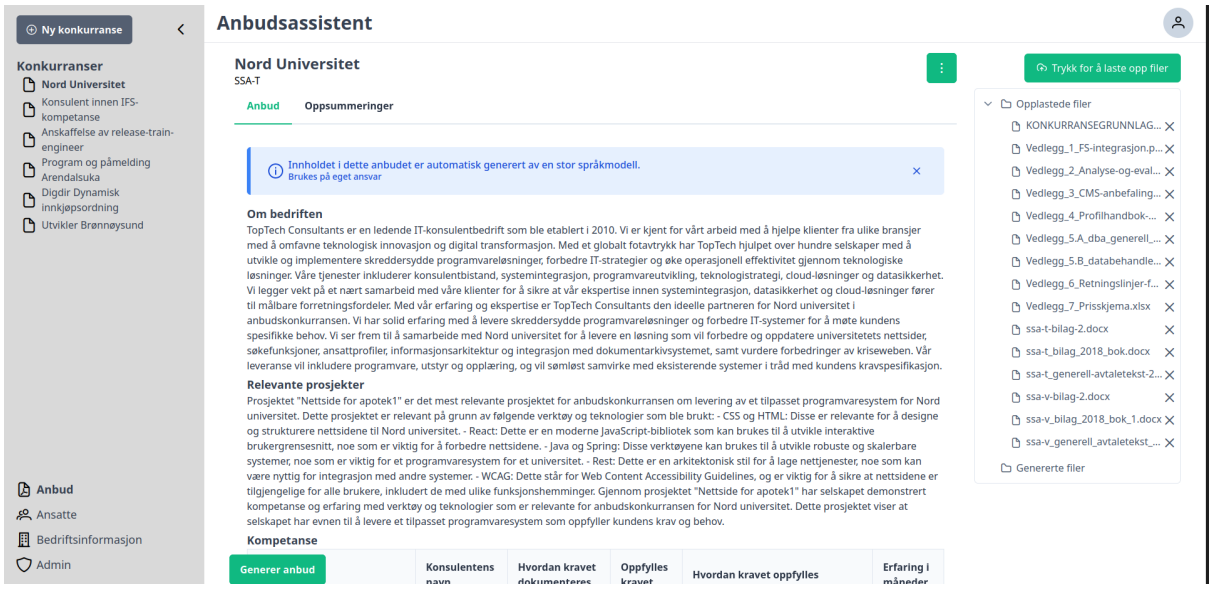
Hjem

Hjemruten er landingssiden til nettportalen, denne er åpen for alle og inneholder innloggingsskjema for brukerne. Når en bruker logger ut, blir man sendt til denne siden.

Figur 5-11 - Hjemmesiden

Konkurranser og anbud

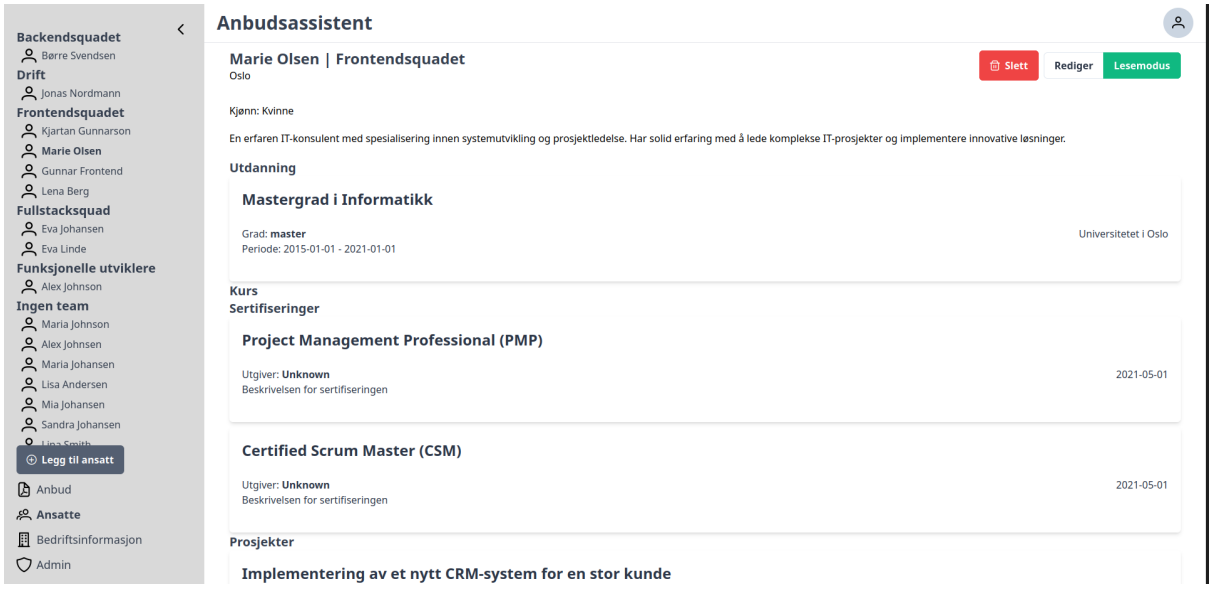
Anbudsgenereringruten den viktigste for applikasjonen, og inneholder en oversikt over alle konkurranser den brukeren eller bedriften har laget, samt mulighet til å laste opp konkurransedokumenter, oppsummere disse eller generere anbud. En vanlig bruker vil kun se sine egne konkurranser, mens en admin-bruker kan se alle for sin egen bedrift.



Figur 5-12 - Side for anbudsgenerering

Konsulenter

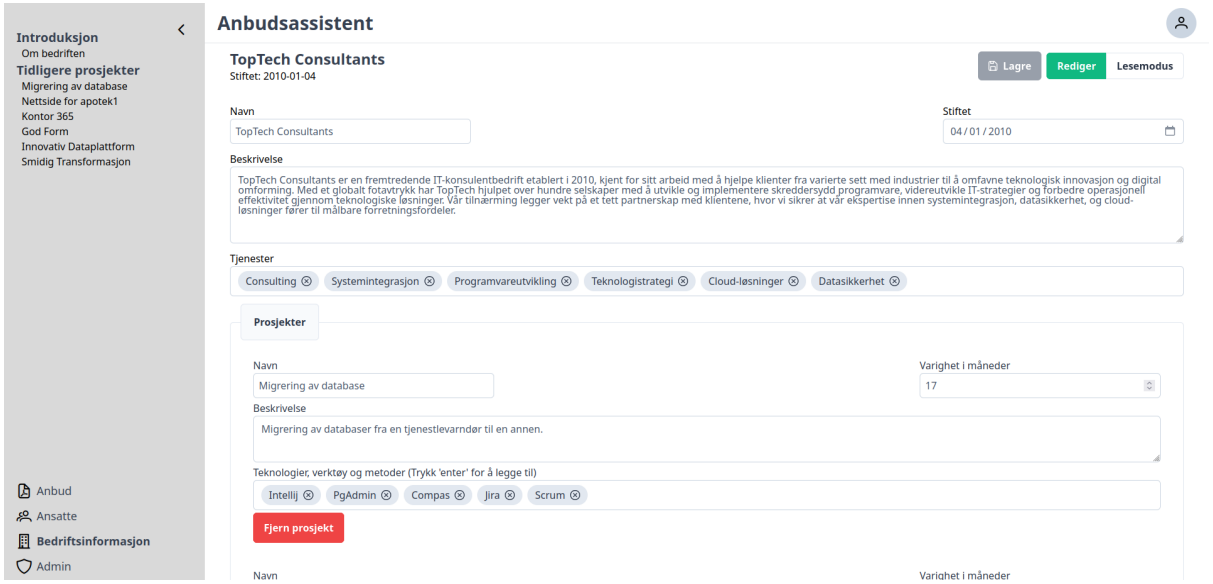
Konsulentruten inneholder en oversikt over alle konsulenter som tilhører bedriften. Her er det også mulig å legge inn nye ansatte, fjerne eller endre dem. Man kan lett legge til nye konsulenter ved å laste opp en CV-fil, og eventuelt legge inn informasjon manuelt. Denne siden er kun tilgjengelig for admin-brukere.



Figur 5-13 - Konsulentoversikt for en valgt konsulent

Bedriftsinfo

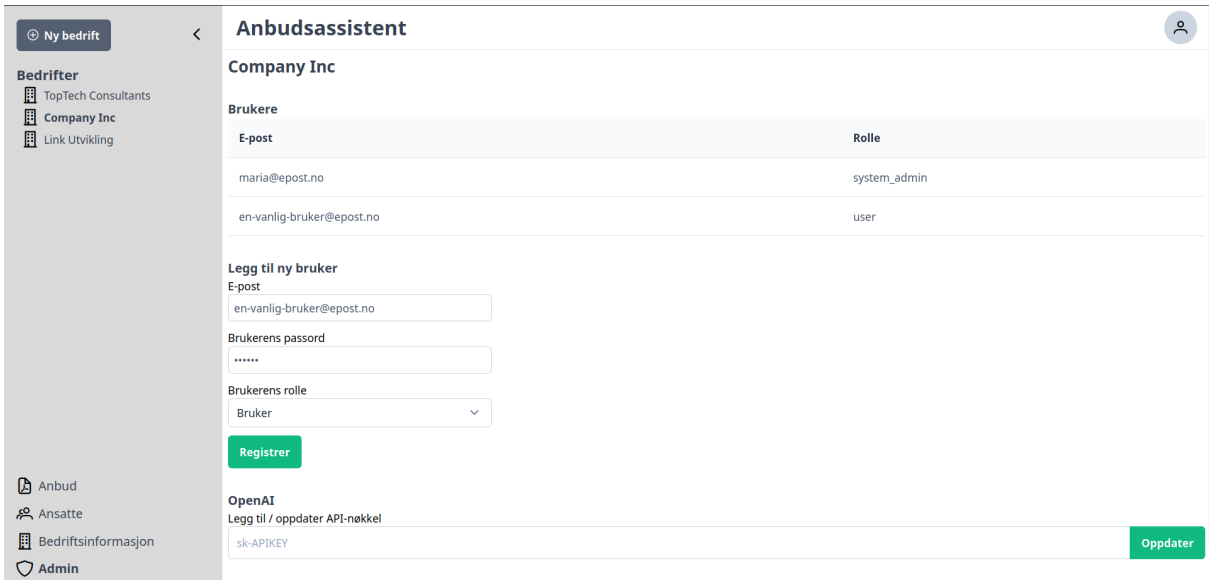
Denne ruten inneholder informasjon om bedriften, her er det mulig å legge inn og endre info om bedriften, som for eksempel sammendrag av tidligere prosjekter som bedriften har utført.



Figur 5-14 - Bedriftsdata for en bedrift

Systemadmin

Den siste ruten er kun åpen for systemadmin-brukere og kan brukes for å opprette nye bedrifter, samt nye brukere og gir en oversikt over disse.



Figur 5-15 - Administratorside for en valgt bedrift

5.5 Bruk av språkmodellene

Til tekstgenerering blir modellen GPT-3.5 Turbo brukt der det er behov for å sende mange API-kall og/eller utføre enkle oppgaver, for eksempel ved uthenting av informasjon fra dokumenter og å lage grove oppsummeringer. GPT-4 Turbo produserer bedre tekst og kan løse mer kompliserte oppgaver, og blir brukt der ny tekst skal genereres, som ved svar på krav, oppsummering av konkurranse og introduksjon av bedrift.

I tillegg blir modellen Text Embedding 3 Small brukt for all tekst-embedding i prosjektet. Det er viktig at det kun brukes én embedding-modell, da hver embedding-modell representerer tekst på litt forskjellig måte, og muligens med forskjellig dimensjon på embedding-vektorene. Ved å bruke kun én embedding-modell, sørger man for at like vektorer har lik betydning. Embedding-modellen benyttes for å generere embedding-vektorer ut fra tekst, som igjen brukes ved vektorsøk i databasen. Den genererte embeddingen lagres sammen med tilhørende tekst i databasen.

5.6 Prompt Engineering

Prompt engineering er et ungt fagfelt i stadig utvikling, og det finnes ingen fasit på den beste prompten for enhver situasjon. Det finnes derimot noen mønstre og fremgangsmåter som har vist seg å være effektive for å produsere gode utdata og minimere risiko for hallusinering. Ettersom API-et til OpenAI er brukt i prosjektet, er det tatt utgangspunkt i OpenAI-dokumentasjonen rundt prompt engineering (OpenAI, 2024d).

5.6.1 Språk

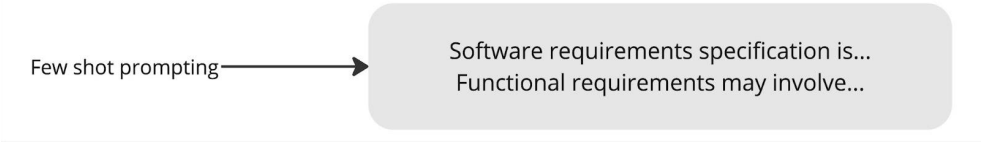
OpenAI sine språkmodeller er trent på enorme mengder data, men mesteparten av treningsdataene er engelskspråklige, ettersom engelsk er det dominerende språket på internett og i en god del andre ressurser som blant annet forskningsrapporter. Derfor vil modellene ha blitt eksponert for mer engelskspråklig tekst under trening, og være bedre på å prosessere og generere engelskspråklig tekst, samt yte bedre når de løser engelskspråklige oppgaver. Det er mulig å lage prompter på engelsk, men spesifisere at utdata fra modellen skal være på norsk, noe som er gjort i dette prosjektet.

5.6.2 Temperatur

Når man setter opp en OpenAI språkmodell, kan man gi med en parameter kalt temperatur. Temperatur er et mål på hvor ofte modellen velger et token som ikke er den mest sannsynlige. Høyere temperatur fører til mer tilfeldige og kreative tekster. I tilfeller som dette, der man benytter modellen til faktuelle brukstilfeller som uthenting av data og svar på spørsmål, er en temperatur på 0 best og er derfor det som er brukt under utvikling av dette prosjektet.

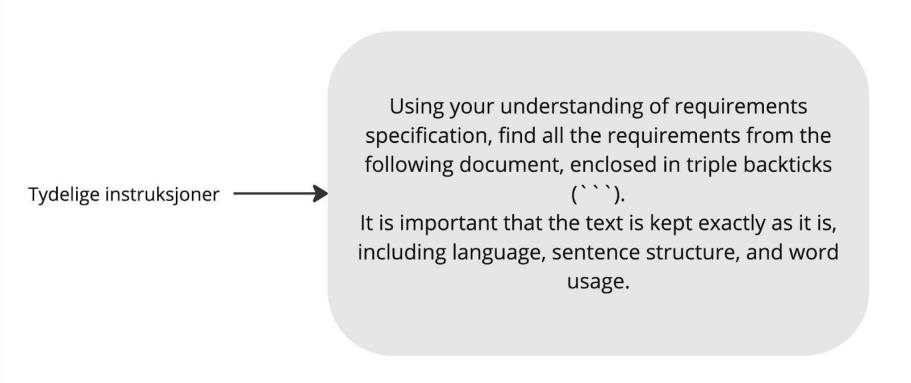
5.6.3 Tydelige instruksjoner

Under utforming av prompten er det lagt stor vekt på å få med alle detaljer i oppgaven som skal utføres, slik at det ikke er noen tvetydighet rundt hva som skal genereres. Dersom prompten handler om begreper som ikke er godt kjent, benyttes "few shot prompting" med definisjon på uvanlige begreper.



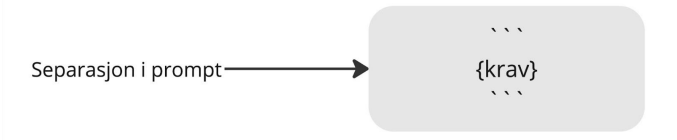
Figur 5-16 - Few shot prompting tydeliggjør oppgaven ved å gi modellen ekstra informasjon.

Det er gunstigere å gi instruksjoner modellen skal utføre, enn å spesifisere uønsket adferd. Gruppen har erfart at negative instruksjoner som "Du skal aldri…" eller "Ikke bruk ordet…" ofte har motsatt effekt. En teori om hvorfor dette skjer er fordi språkmodeller gjør en sannsynlighetsbasert prediksjon, der sannsynligheten er vektet ut fra alle ordene i inndataen. For oss mennesker vil negasjon ha svært sterk innflytelse på forståelsen av teksten, men for en maskinlæringsmodell kan negasjon ha mye mindre innvirkning. En mulig grunn til at negasjon vektes for lavt i store språkmodeller kan være at treningssettet inneholder mer positive enn negative instruksjoner (Truong *et al.*, 2023, s. 101–102). Derfor er det for det meste brukt positive instruksjoner i promptene. Promptene inneholder i enkelte tilfeller instruksjoner om hvilket språk svaret skal være på, og hvilken lengde av svar som forventes. Det er også forsøkt å lage så kortfattede, men detaljerte instruksjoner som mulig.



Figur 5-17 - Tydelige, positive instruksjoner brukes for å spesifisere ønsket adferd.

Modellene har lettere for å skjønne separasjoner i prompten hvis det blir brukt skilletegn for å tydelig markere distinkte deler av inndataen. I tilfeller der prompten handler om å jobbe med data, blir dataene lagt inn med tre "backticks" (```) på hver side.



Figur 5-18 - Tydelig separasjon i prompt markerer når viktig informasjon starter og slutter.

5.6.4 Formatering

Utdata fra modellen skal ofte lagres i databasen på et bestemt format, og det er derfor ønskelig å få ut svaret fra modellen på dette formatet. Der det har vært nødvendig med spesifikk formatering, har biblioteket Pydantic blitt tatt i bruk sammen med LangChain sin "JsonOutputParser" og GPT-modellene sin "JSON-modus", noe som tvinger utdata til å være på JSON-format.

JsonOutputParser tar inn et Pydantic-objekt og kan generere formateringsinstruksjoner fra denne. Når LangChain-prompten settes opp, gis formateringsinstruksjonene med som en partiell variabel i prompten. Figur 5-19 viser denne prosessen fra en enkel Pydantic klasse, til formateringsinstruksene i form av JSON, som settes inn i prompten i "{format_instructions}" feltet.

Det er også mulig å legge inn ekstra informasjon til hvert felt i Pydantic-modellene, som en beskrivelse av det som skal fylles inn, eksempler på data som kan fylles ut og en standard verdi dersom modellen ikke finner noe. Dette gjør det lett å kontrollere nøyaktig hvilken informasjon som blir fylt inn i hvert felt, og gjør prompten mer oversiktlig (Figur 5-20).

Figur 5-19 - Formateringsinstruksjoner genereres fra Pydantic-klasser og settes inn i prompten.

```python
class Project(BaseModel):
    name: str = Field(description="Navnet på prosjektet")
    description: str = Field(description="En grundig beskrivelse av prosjektet", default=None)
    duration: str = Field(description="Varigheten av prosjektet i måneder. Kun heltall", default=None, example="12")
    roles: list[str] = Field(description="Rollene i prosjektet", default=None)
    tools: list[str] = Field(description="En liste over teknologier, verktøy og metoder brukt i prosjektet",
                             default=[], examples=["Python", "Django", "React", "Jira", "Git", "Scrum"])
```

Figur 5-20 - Pydantic-modell med instrukser.

5.7 Generering av anbud

Dette kapittelet gjennomgår de ulike delene av anbudsgenereringen. Generering av anbud er et sammensatt problem som krever mange forskjellige metoder. For å illustrere flyten i hver del av anbudsgenereringen blir det lagt ved diagrammer. Diagrammene er fargekodet ut fra fargene i Figur 5-21.

Figur 5-21 - Fargekoding av forskjellige ansvarsområder i applikasjonen

5.7.1 Oppsummering av konkurranse

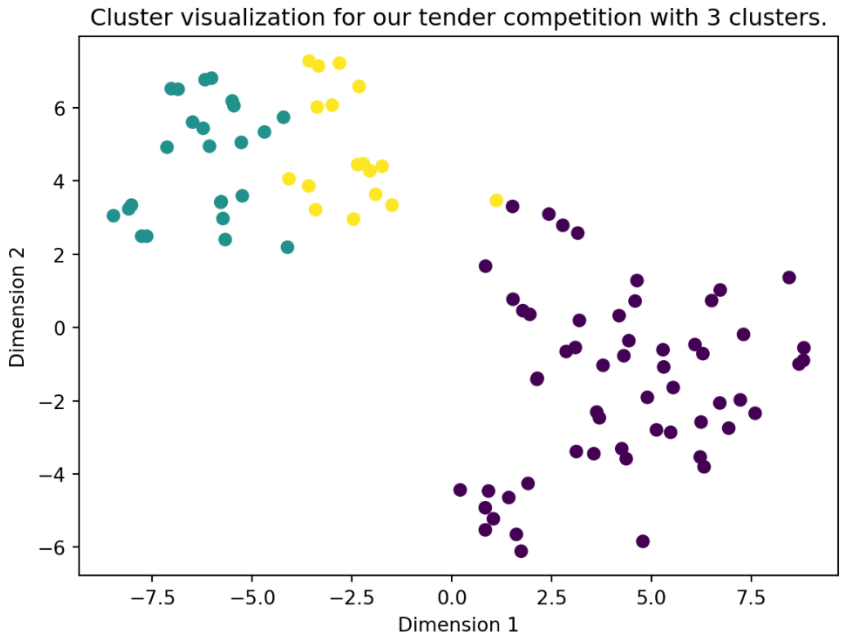
Oppsummering av anbudskonkurransen er nyttig for brukeren av systemet, da man raskt kan se om konkurransen er interessant å levere anbud på. Et eksempel på en oppsummering av en anbudskonkurranse kan sees i Figur 5-22.

Figur 5-22 - Oppsummering av en anbudskonkurranse for Nord Universitet

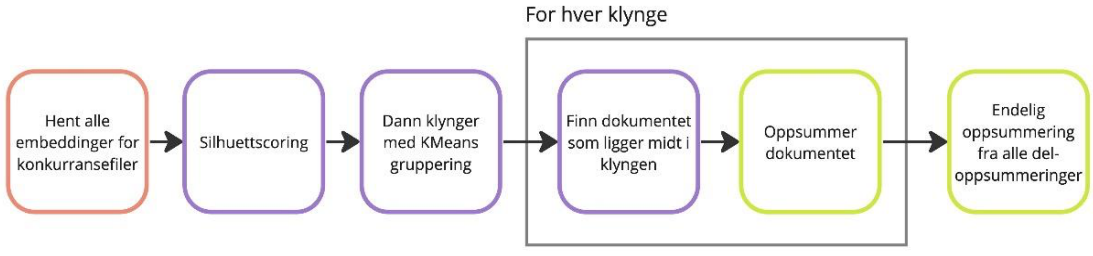
Siden anbudskonkurranser kan inneholde vilkårlig mange dokumenter, er det ønskelig å begrense mengden inndata-tokener slik at oppsummeringen blir mindre kostbar. Dette gjøres med *K-Means gruppering*, en metode for å dele dokumentbiter inn i *klynger* med sammenhengende tema (Figur 5-23). I denne algoritmen er antall klynger forhåndsbestemt, kalt *k*. Til å begynne med blir *k* tilfeldige datapunkter satt som senter i hver sin klynge. Deretter går algoritmen gjennom en løkke der alle datapunkter tilordnes klyngen til det nærmeste senteret, for så å kalkulere et nytt senter ved å finne gjennomsnittet av alle datapunktene. Denne løkken fortsetter helt til klyngene ikke endrer på seg lengre. For å finne den optimale verdien for *k,* altså hvor mange klynger man burde ha, brukes en metode kalt silhuettscoring. Dette er en måling på hvor like datapunkter er til sin egen klynge (*samhold*) i forhold til andre klynger (*separasjon*). Hvert datapunkt får en verdi der –1 betyr at datapunktet har stor separasjon fra sin egen klynge, og 1 betyr at datapunktet har høyt samhold med sin klynge. Dersom de fleste datapunkter i en klynge har høy verdi, anses klyngene å være gode. Silhuettscoring lager altså grupperinger med ulik mengde klynger, og gir en score for hver av dem. Deretter brukes

verdien for *k* med tilhørende høyest score. For å utføre K-Means gruppering og silhuettscoring er maskinlæringsbiblioteket Scikit-learn blitt brukt (Scikit Learn, 2024a kap. 2.3.2, 2024b).

Siden klynger etter K-Means gruppering burde inneholde dokumentbiter som handler om det samme, velges den dokumentbiten som er i midten av hver klynge som en representativ bit for den klyngen. De representative bitene sendes så inn i en avbildningsreduksjon-kjede (Map Reduction Chain). En slik kjede genererer en oppsummering (*avbildning*) for hver bit, for så å generere en ferdig oppsummering ut fra alle oppsummeringene (*reduksjon*). Hele prosessen for oppsummering er illustrert i Figur 5-24.



Figur 5-23 - Visualisering av klynger for en anbudskonkurranse. Her med tre klynger. Vektorrommet er i denne visualiseringen redusert fra 1536 til 2 dimensjoner (Kristiansen og Vatnelid, 2024b).



Figur 5-24 - Flyt for å generere oppsummering av anbudskonkurranse

5.7.2 Oppsummering av avtaletekst

For å lage et sammendrag av avtaleteksten, som inneholder bestemmelser og lignende fra konkurransen, finnes først riktig fil ved hjelp av vektorsøk. Ut fra om konkurransen er SSA-B eller SSA-T velges så en kapitteloversikt som mates til modellen for å trekke ut viktig informasjon relatert til hvert kapittel i avtaleteksten. Denne kapitteloversikten er hentet fra SSA-B og SSA-T sine maler for avtaletekst (Doffin, Database for offentlige anskaffelser, 2024c), og inneholder informasjon om viktige punkter fra hvert kapittel, for eksempel informasjon om mislighold (Figur 5-25).

- **Betalingsmislighold:** Hvis forfalt vederlag med tillegg av forsinkelsesrenter ikke er betalt innen 30 (tretti) kalenderdager fra forfall, kan Leverandøren sende skriftlig varsel til Kunden om at avtalen vil bli hevet dersom oppgjør ikke er skjedd innen 60 (seksti) kalenderdager etter at varselet er mottatt. Heving kan ikke skje hvis Kunden gjør opp forfalt vederlag med tillegg av forsinkelsesrenter innen fristens utløp.
- **Prisendringer:** Timepris for tjenester kan endres ved hvert årsskifte tilsvarende økningen i Statistisk sentralbyrås konsumprisindeks (hovedindeksen), første gang med utgangspunkt i indeksen for den måned.

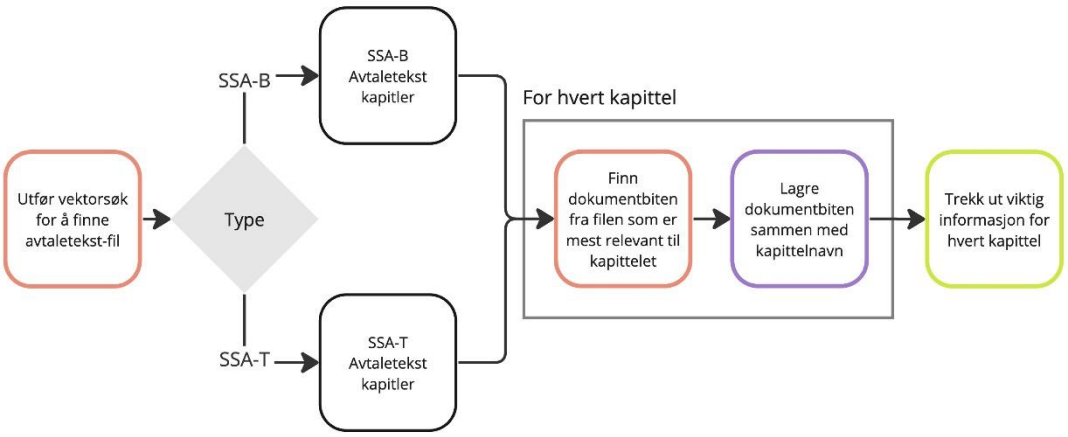
Figur 5-25 - Utsnitt fra oppsummering av avtaletekst

For hvert kapittel benyttes vektorsøk for å finne de to mest relevante dokumentbitene, disse blir lagret sammen med kapittelnavnet i en *ordbok* (dictionary). Til slutt sendes ordboken til en Chains-klasse der det blir gjort kall til språkmodellen med navnet på kapittelet og de relevante dokumentbitene, sammen med formateringsinstruksjoner ut fra en modell (Figur 5-26). For å gjøre denne prosessen raskere sendes kallene til GPT-3.5 Turbo asynkront og resultatene samles til slutt (Figur 5-27).

```
class AvtaletekstSummary(BaseModel):
    tittel: str = Field(description="The title of the chapter")
    info: str = Field(description="Important information from the chapter that the contractor should know")


class AvtaletekstSummaries(BaseModel):
    points: list[AvtaletekstSummary]
```

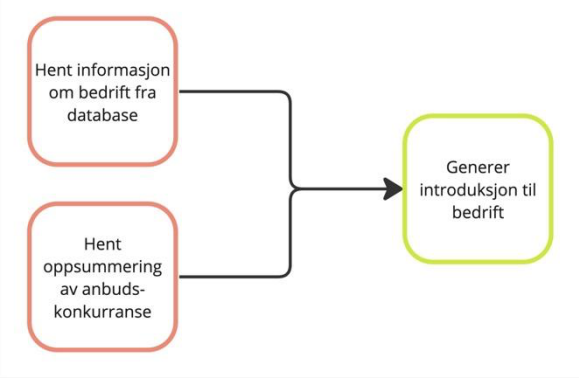
Figur 5-26 - Modell for sammendrag av avtaletekst



Figur 5-27 - Flyt for å hente ut viktig informasjon fra avtaletekst

5.7.3 Introduksjon av bedrift

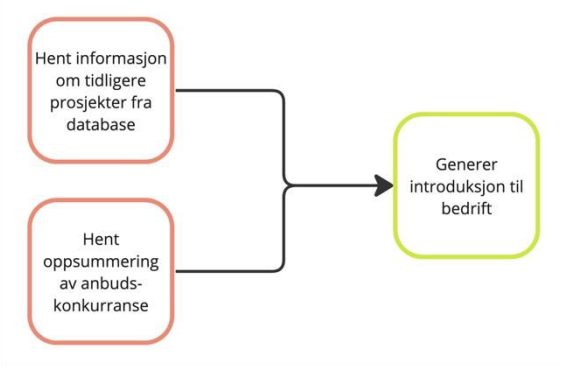
Det er vanlig å inkludere en introduksjon av bedriften. Bedriften har selv lagt inn relevant informasjon i systemet. Det er ønskelig å spisse introduksjon av bedriften mot anbudskonkurransen, for å gi et inntrykk av at bedriften er den rette for oppdraget. Dette gjøres ved å hente inn oppsummering av anbudskonkurransen, for så å sende både oppsummering og bedriftsinformasjon til GPT-3.5 Turbo (Figur 5-28).



Figur 5-28 - Flyt for å generere introduksjon av bedrift

5.7.4 Relevante tidligere oppdrag

Det er også vanlig å legge ved informasjon om relevante tidligere oppdrag som bedriften har utført. Dette er et kort avsnitt som beskriver en eller flere prosjekter som kan være ganske like til konkurransen og derfor interessant å ta med. For SSA-T sender man bedriftens tidligere prosjekter til språkmodellen. For SSA-B sender man de valgte konsulentene sine prosjekter i stedet. Likt for begge er at det blir også sendt en oppsummering av anbudskonkurransen i tillegg, som gjør at modellen kan velge de mest relevante prosjektene å ta med. (Figur 5-29).



Figur 5-29 - Flyt for å generere tekst om tidligere relevante prosjekter

5.7.5 Valg av kompetanse

I et anbud legger man vanligvis ved informasjon om konsulenter som skal utføre oppdraget eller leies ut, avhengig av hvilken type konkurranse det er. Konsulentdataene legges vanligvis med i en kompetansematrise (Figur 5-30), som inneholder krav som skal oppfylles, navn på konsulentene og hvordan kravet oppfylles. Det er opp til språkmodellen å vurdere om kravet er oppfylt, og dersom det ikke er det, vil flere av feltene holdes tomme. Hvordan kravet oppfylles er delvis beskrevet i form av en punktliste som er lett å lese, og inneholder viktig informasjon som relevante sertifiseringer og prosjekter. Hvordan denne kompetansematrisen blir laget og hvordan man finner konsulenter varierer ut fra om konkurransen bruker avtalemalen SSA-B eller SSA-T.

Kompetanse

Krav	Konsulentens navn	Hvordan kravet dokumenteres	Oppfylles kravet	Hvordan kravet oppfylles	Erfaring i måneder
Konsulenten skal ha minimum 5 års erfaring med å innføring- og bruk av smidig metode.	Marie Olsen	PROSJEKT, ERFARING, SERTIFIKAT	Ja	• Sertifisert som Certified Scrum Master (CSM). • Implementering av CRM-system med bruk av Agile-metodikk. • Utvikling av mobilapplikasjon med Agile-metodikk. Marie har jobbet med smidig metodikk i flere prosjekter, inkludert implementering av et nytt CRM-system og utvikling av en mobilapplikasjon.	60

Figur 5-30 - Utsnitt fra kompetansematrise

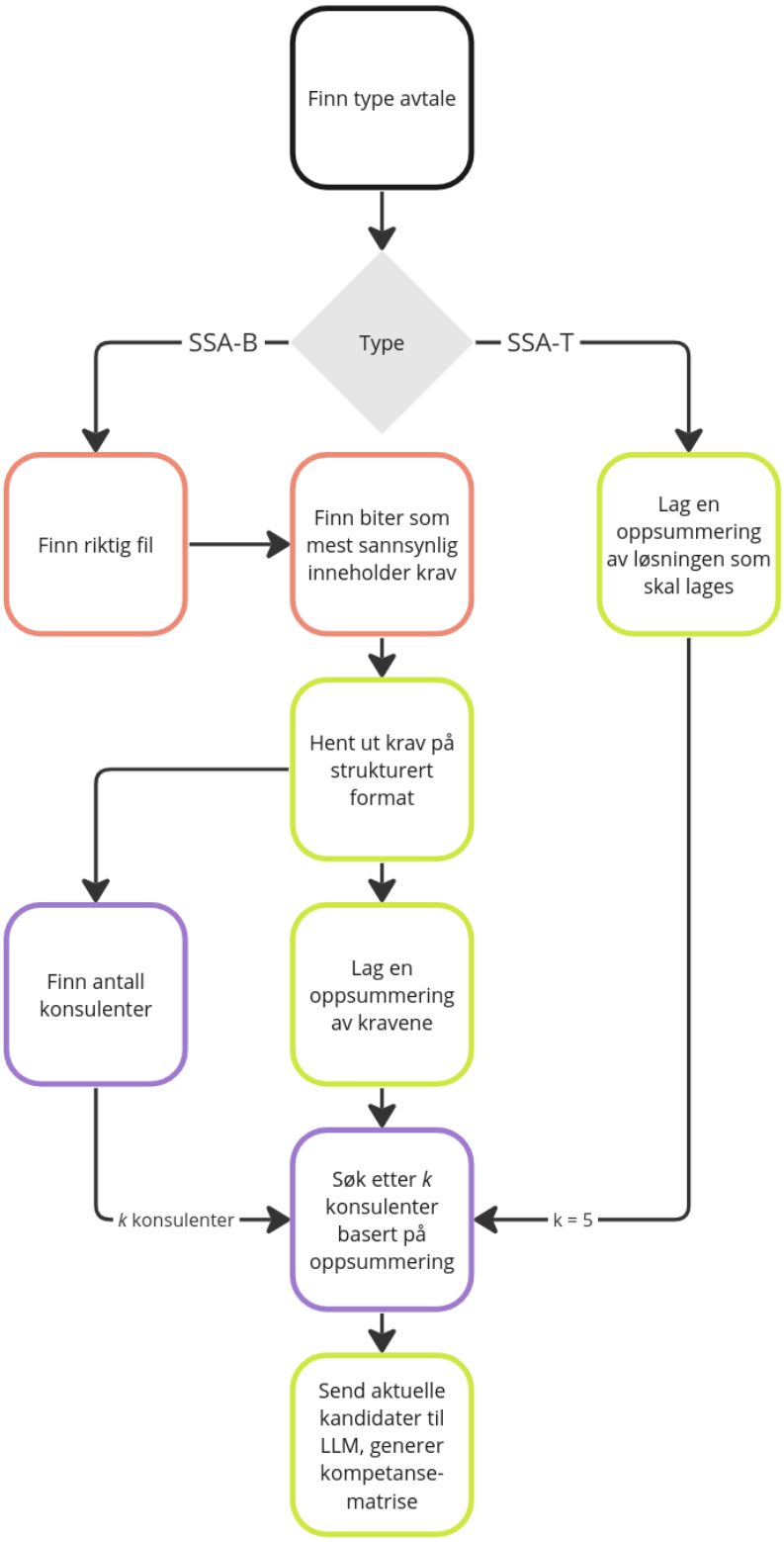
Bistandsavtalen (SSA-B)

I en bistandsavtale er kompetansen det viktigste med leveransen. Det skal derfor fremlegges gode beskrivelser av konsulentene, med informasjon om hvor relevant kompetanse kommer fra (utdanning, kurs, tidligere prosjekter og lignende). For å finne kompetanse i SSA-B anbudskonkurranser, starter man med å finne riktig fil og relevante dokumentbiter fra denne filen ved hjelp av vektorsøk. Deretter brukes GPT-3.5 Turbo for å hente ut krav til kompetanse fra dokumentbitene og returnere disse på et strukturert JSON-format.

Før man søker blir det laget en oppsummering av alle kravene til konkurransen, som blir embeddet og brukt for å søke etter kompetansen. Hvor mange konsulenter man søker etter, må først finnes og skal være tydelig spesifisert i konkurransedokumentene, som for eksempel "1 rolle" eller "1 fulltidsekvivalent (FTE)". Da blir det først gjennomført et vektorsøk for å finne de mest relevante bitene, som blir matet inn i språkmodellen for å finne antallet som konkurransen ber om. Deretter blir kravene og de aktuelle konsulentene sendt til språkmodellen, hvor den lager en kompetansematrise basert på kravene og tilgjengelig kompetanse (Figur 5-31).

Utviklings- og tilpasningsavtalen (SSA-T)

I SSA-T anbudskonkurranser har man ikke alltid krav til kompetanse. Bedriften som leverer anbudet beskriver løsningen og må finne konsulenter som kan bistå med utvikling. Det er ofte vanlig å legge ved en oversikt over konsulenter som har nødvendig kompetanse her også. Siden det ikke er konkrete krav i SSA-T, lages en oppsummering av anbudskonkurransen med fokus på de tekniske aspektene ved det som skal utvikles. Denne oppsummeringen genereres på samme måte som oppsummering av anbudskonkurransen, beskrevet i 5.7.1. Denne oppsummeringen blir embeddet, og brukes til å utføre et vektorsøk etter fem konsulenter. Til slutt blir oppsummeringen og konsulentdataene sendt til GPT-3.5 Turbo, som deler oppsummeringen inn i forskjellige "krav" og tilordner en av konsulentene til kravet (Figur 5-31). Dette returneres i et strukturert JSON-format.



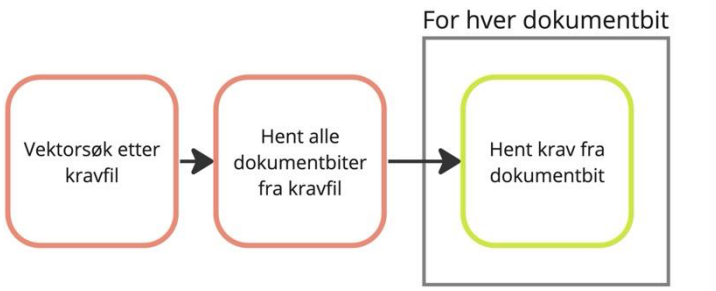
Figur 5-31 - Valg av konsulenter til kompetansematrise

5.7.6 Hente ut og svare på krav for SSA-T

Ekstraksjon av krav byr på flere utfordringer, blant annet at det ikke er noe gitt format på kravene, verken i fremstilling eller innhold, og at det eksisterer "krav-lignende" setninger i konkurransedokumentene som ikke egentlig er krav. For å forsøke å forhindre at "krav-lignende" setninger blir tatt med, er promptene spisset slik at modellen skal finne riktig data, ved å blant annet gi eksempler på hvordan et krav ser ut. Siden anbudskonkurranser kommer på forskjellig format med ulik oppsett av kravspesifikasjon, er det fortsatt noen ikke-krav som blir tatt med. Det er ikke hensiktsmessig å spisse prompten for mye, siden innholdet i konkurransene kan variere, og en for spisset prompt vil føre til at reelle krav blir utelatt.

Før man kan søke i en fil etter krav må man finne riktig fil, uten at det skal være nødvendig for en bruker å markere hvilken fil som inneholder krav. Dette gjøres ved vektorsøk etter vanlige ord og temaer i en slik kravfil, der de *k* nærmeste bitene hentes, med tilhørende score og filnavn, og det filnavnet med gjennomsnittlig høyest score velges.

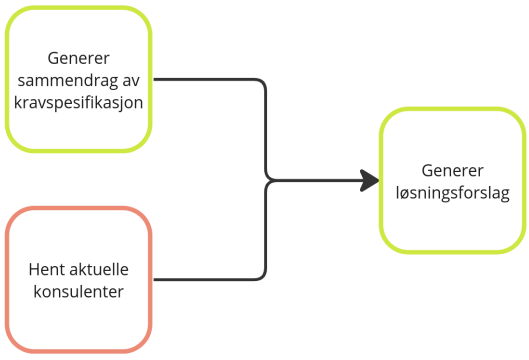
Kravene trekkes så ut av dokumentbitene ved asynkrone kall til GPT-3.5 Turbo (Figur 5-32), og returneres på et strukturert JSON-format.



Figur 5-32 - Flyt for uthenting av krav

5.7.7 Beskrivelse av løsningen som skal utvikles for SSA-T

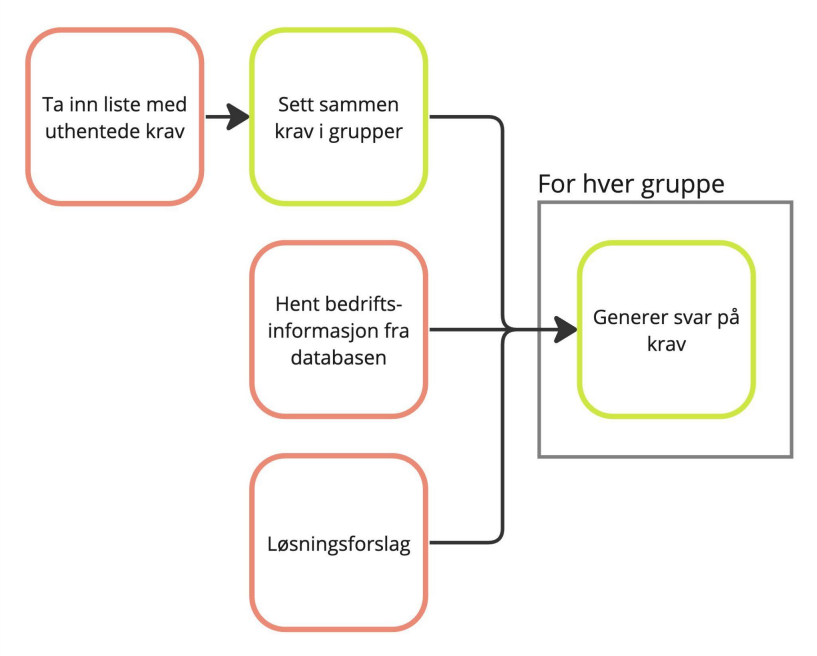
En sentral del av anbudet er å gi forklaring på hvordan løsningen skal utvikles. Dette gjøres ved å lage en PromptTemplate som tar inn alle krav fra konkurransen, og ber GPT-4 Turbo lage en teknisk oppsummering av løsningen som skal lages. Denne oppsummeringen blir sendt sammen med de aktuelle konsulentene til GPT-4 Turbo, som gir en løsningsbeskrivelse tilbake (Figur 5-33).



Figur 5-33 - Flyt for generering av løsningsforslag

5.7.8 Svar på krav for SSA-T

Etter man har hentet ut alle kravene til en anbudskonkurranse, er det nødvendig å generere svar på dem. Det blir gjort ved at man tar inn løsningsforslaget og data om bedriften, og sender flere asynkrone API-kall til GPT-4 Turbo modellen med løsningsforslag, bedriftsdata og krav på hvert API-kall. For å begrense mengden API-kall, slår man sammen flere krav i grupper og ber modellen svare på en gruppering av krav om gangen (Figur 5-34).



Figur 5-34 - Flyt for å generere svar på krav

6 RESULTATER

6.1 Evalueringsmetode

Fokus under utvikling har vært å utvikle en tjener som kan generere utkast til anbud, og forbedre denne så mye som mulig. Klientsiden til applikasjonen er derfor enkel, og brukes i hovedsak til brukertesting, som er hovedmetoden for evaluering av kvalitet på utkast til anbud. I dette kapittelet gjennomgås de forskjellige evalueringsmetodene benyttet i prosjektet.

6.1.1 Brukertest

Hver brukertest vil gjennomføres i to ulike faser. Den første fasen er selve testingen av applikasjonen, mens den andre fasen er at testeren svarer på et skjema med spørsmål rundt applikasjonens brukbarhet og kvalitet.

Under testing av applikasjonen vil testerne få tilgang til systemet og få litt tid til å utforske de forskjellige delene av nettportalen. Deretter vil testerne få mulighet til å laste opp filer i systemet og generere et utkast til anbud. Under brukertesten vil det gis minimalt med veiledning for å få et bedre inntrykk av hvordan en person som ikke er kjent med systemet samhandler med det. Underveis i testen vil en av gruppens medlemmer notere eventuelle kommentarer eller problemer testeren støter på.

Etter gjennomført test skal testeren svare på en spørreundersøkelse. Brukertestingen vil blant annet benytte et *SUS-skjema*. SUS-skjema (*System Usability Scale*) er et skjema med ti punkter som måler brukerens subjektive erfaring med et system (Brooke, 1995). Skjemaet inneholder spørsmål med både negativ og positiv ordlyd i spørsmålene, for å motvirke bias eller skjevhet i svar fra brukere som følge av at de ikke tenker nøye gjennom spørsmålene de svarer på. Ved å veksle mellom positiv og negativ ordlyd må respondentene lese spørsmålet nøye og tenke gjennom om de er enig med utsagnet eller ikke. Alle spørsmål har en skala på fem mulige svar, fra helt enig til helt uenig.

Det er utarbeidet 6 ekstra spørsmål i samme stil som SUS-skjemaet, rettet mer spesifikt mot brukbarheten av anbudsgenerering i systemet. Disse spørsmålene har også vekslende negativ og positiv ordlyd og en skala på fem mulige svar. Spørsmålene er utarbeidet for å evaluere om testpersonene mener at systemet er nyttig og sparer dem tid.

Et utfylt skjema vil gi et tall som representerer brukbarheten til systemet. Hvert spørsmål har en verdi på mellom 1 og 5, ut fra hva testeren har krysset av for. Etter poengberegning gir hvert spørsmål mellom 0 og 4 poeng, der spørsmålene som er positivt ladet gir *verdi - 1*, og spørsmålene som er negativt ladet gir *5 - verdi* poeng. Poengene summeres og ganges med 2.5 for å få den endelige poengsummen for skjemaet. SUS-skjemaet har dermed en poengsum mellom 0 og 100, der 100 er best. De 6 ekstra spørsmålene har en poengsum på mellom 0 og 60, der 60 er best.

Tabell 6-1 - SUS-skjema

Nummer	Spørsmål	Poengberegning
1	I think that I would like to use this system frequently.	Verdi - 1
2	I found the system unnecessarily complex.	5 - Verdi
3	I thought the system was easy to use.	Verdi - 1
4	I think that I would need the support of a technical person to be able to use this system	5 - Verdi
5	I found the various functions in this system were well integrated.	Verdi - 1
6	I thought there was too much inconsistency in this system.	5 - Verdi
7	I would imagine that most people would learn to use this system very quickly.	Verdi - 1
8	I found the system very cumbersome to use.	5 - Verdi
9	I felt very confident using the system.	Verdi - 1
10	I needed to learn a lot of things before I could get going with this system.	5 - Verdi

Tabell 6-2 - Brukbarhet av anbudsgenerering-tjenesten

Nummer	Spørsmål	Poengberegning
1	I am happy with the generated output.	Verdi - 1
2	I don’t think the generated output from the system is usable.	5 - Verdi
3	I feel that this system saves me time.	Verdi - 1
4	I would rather do this process manually.	5 - Verdi
5	I think the system works as expected.	Verdi - 1
6	I experienced many problems while using the system.	5 - Verdi

I tillegg vil noen ekstra spørsmål bli besvart, som har med kvalitet og stil på spesifikke deler av det genererte anbudet. Spørsmål 1-5 benytter en poengskala fra 1 – Svært dårlig til 5 – Svært bra, mens spørsmål 6 er flervalg.

Tabell 6-3 - Kvalitet på utdata

Nummer	**Spørsmål**
1	Kvalitet på sammendrag av avtaletekst (omfang av avtale, viktige punkter og lignende)
2	Kvalitet på introduksjon av bedriften (om bedriften og relevante tidligere prosjekter)
3	Kvalitet på sammendrag av konkurranse (hva skal produseres/leveres)
4	Kvalitet på valgte kandidater og begrunnelse av disse valgene (kompetansematrise)
5	Kvalitet på svar på kravspesifikasjon
6	Hvordan vil du beskrive personligheten til den genererte teksten? - For robotisk - teksten virker mekanisk og/eller høres ikke ut som et menneske har skrevet den - For profesjonell - teksten føles overdrevent profesjonell og distansert - For uprofesjonell - teksten er for uprofesjonell for formålet - Riktig - teksten er skrevet omtrent slik jeg ønsker den - Annet

6.1.2 Testing av utdata

I all hovedsak er testingen av utdata manuell, både ved brukertesting (Tabell 6-2) og internt i gruppen ved å sammenligne utdata opp mot bedriftsdata og konkurransedokumenter. Det kan være nyttig å automatisere denne prosessen, ved å bruke en metode for likhetstesting.

En mulig løsning for testing av utdata er *BERTScore*, som benytter embedding-modeller for å sammenligne tekster (Zhang *et al.*, 2020). Ved å benytte embeddinger for å sammenligne resultatene kan man verifisere at to tekster handler om det samme selv om de er ulike. Dette gjøres ved å sende med en eller flere referansetekster som er teksten man ønsker å sammenligne med, og en kandidattekst som er resultatet man får fra språkmodellen. BERTScore gir oss tre ulike metrikker som man kan bruke for å sammenligne resultater: *presisjon*, *sensitivitet* (recall) og en *F1 score*. Presisjon forteller oss hvor likt hvert ord i kandidatteksten er til et av ordene i referanseteksten. Sensitivitet forteller oss hvor bra kandidatteksten fanger opp det referanseteksten handler om. F1 score er et harmonisk gjennomsnitt av presisjon og sensitivitet, og gir en enkelt poengsum som balanserer begge metrikkene. Man kan også legge på en valgfri

vekting av viktighet, hvor sjeldne ord øker poengsummen mer enn andre vanlige ord. Dette blir gjort ved å sende med en eller flere dokumenter som blir brukt for å lage vekttabellen ut fra hvor mange av dokumentene hvert ord opptrer i. Disse vektene kalles "*inverse document frequency*" (idf), og defineres før man kjører testen. Hvor mye det utgjør kommer an på dokumentene modellen blir gitt før kjøring, i noen tilfeller har det nesten ingen innvirkning på resultatet. Figur 6-1 viser flyten for utregning av sensitivitet med idf-vekting ved bruk av BERTScore.

For å kalkulere sensitivitet for en tekst blir hver token i referanseteksten (x) multiplisert med den token i kandidatteksten ($\hat{x}$) som har høyest cosinuslikhet med den aktuelle token fra referanseteksten. Deretter blir resultatet for hver token summert, og delt på antall tokener i referanseteksten.

$$R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} max(\hat{x}_j \in \hat{x})\, x_i^T\, \hat{x}_j$$

Dersom man bruker idf-vekting i kalkulering av sensitivitet blir cosinuslikhet kalkulert på samme måte. Deretter blir resultatet for hver token multiplisert med idf-vektingen for den tokenen, før de vektede resultatene summeres. Til slutt blir denne vektede summen delt på den idf-vektede summen av alle tokener i referanseteksten. Ved utregning av presisjon bytter kandidattekst og referansetekst plass i funksjonene.

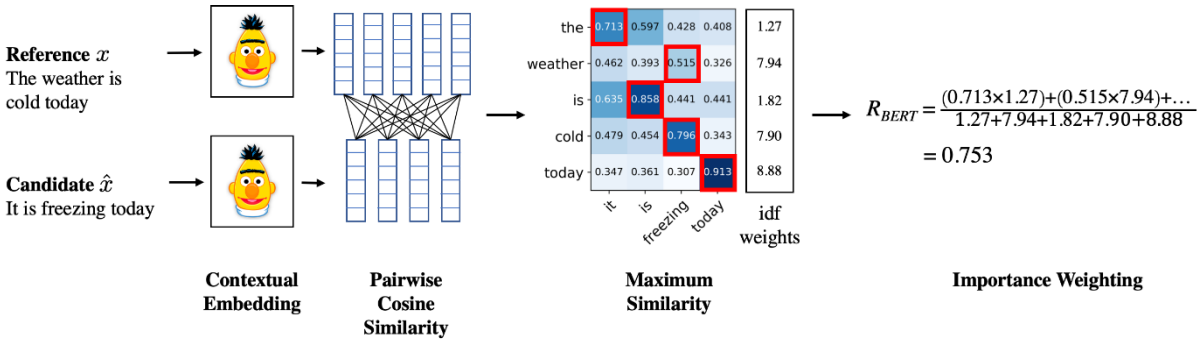
$$R_{BERT} = \frac{\sum_{x_i \in x} max(\hat{x}_j \in \hat{x})\, x_i^T\, \hat{x}_j}{\sum_{x_i \in x} idf(x_i)}$$

En annen mulig løsning for å evaluere utdata er OpenAI Evals (Ziv og Anadkat, 2024). Dette er en offisiell løsning av OpenAI for evaluering av store språkmodeller. Denne løsningen bruker et forventet svar for å verifisere at utdata er riktig. For eksempel kan man evaluere at en modell genererer riktig kode basert på en prompt. Dette vil ikke være aktuelt for vår løsning, siden det er mye tekstgenerering og man kan ikke alltid forutse nøyaktig hva resultatet skal være på forhånd.

Metoden *BLEU* (BiLingual Evaluation Understudy) bruker referansetekst for å kunne gi en evaluering. Referanseteksten i dette tilfellet blir ansett som det riktige svaret, og resultatene som skal bli evaluert sammenlignes opp i mot referansen (Papineni *et al.*, 2002). BLEU benytter *n-grams*, en kontinuerlig sekvens av ord, og sammenligner overlapp mellom referansetekst og resultattekst. Resultatet er en presisjonspoengsum på mellom 0 og 1, der 1 er identiske tekster. BLEU er en relativt enkel testmetode for tekst, og vil ikke ta hensyn til synonymer, semantikk og parafraser.

Valgt løsning for testing av utdata er BERTScore, da den er bedre egnet til å teste semantisk likhet enn de andre. Modellen som skal brukes er Microsoft sin *deberta-xlarge-mnli* modell, siden denne foreløpig har høyest poengsum sammenlignet med menneskelig evaluering (Tianyi, 2024). OpenAI Evals passer best dersom man har et konkret svar klart på forhånd, noe som ikke er aktuelt for dette prosjektet, og BLEU tar ikke hensyn til semantisk likhet. Ved å benytte en slik metode for testing, er det viktig å på forhånd definere en akseptabel grense for

likhet, slik at man ikke blir påvirket av resultatene. I dette prosjektet har en grenseverdi for akseptabel likhet blitt satt til en F1 score på 60%, da tekstene kan være ganske ulike, men fortsatt riktige.



Figur 6-1 - Utregning av BERTScore (Zhang et al., 2020)

6.2 Evalueringsresultat

6.2.1 Brukertester

Første brukertest (Internt)

Før første brukertest var klientsiden omtrent ferdig utviklet, med funksjonalitet for å opprette anbud, laste opp filer, legge inn informasjon om bedrift og lese inn CV for ansatte. Ved generering av anbud ble en kort oppsummering av anbudskonkurransen gitt, sammen med svar på funksjonelle krav i konkurransen. De interne brukerne ga tilbakemelding om at de var svært fornøyde med hvor langt prosjektet var kommet, og at produktet var lett å ta i bruk. Imidlertid savnet de mer av tekstene som skal inn i et anbud, som blant annet en oppsummering av bedriften som leverer anbudet, og oversikt over kompetansen for anbudet i en kompetansematrise. Det ble også uttrykt et ønske om et sammendrag av vilkår og betingelser i konkurransen, slik som dagsbøter, bindingstider og lignende.

Andre brukertest (Internt)

Andre interne brukertest handlet kun om kvalitet på generert utdata. Før denne testen var mer av tekstgenereringen på plass, herunder introduksjon av bedriften, tidligere relevante prosjekter, kompetansematrise, og sammendrag av vilkår og betingelser. Denne brukertesten ble gjennomført med kun én tester, som gikk gjennom genererte svar for en anbudskonkurranse i detalj. Fra denne testen fikk vi gode tilbakemeldinger på kvalitet av generert anbud og hva som burde være fokusområde videre. Blant annet ble det gitt tilbakemelding om at kompetansematrisen burde inneholde informasjon om hvor konsulenten har relevant kompetanse fra (oppdrag, utdanning eller lignende), og at det burde bli laget et sammendrag av foreslått løsning før svar på krav fra kravspesifikasjonen kommer. Testeren var fornøyd med sammendrag og introduksjon av bedrift, men følte at svar på krav lovte litt for mye og gjorde at en løsning produsert ut fra disse kravbesvarelsene ville bli unødvendig stor.

Tredje brukertest (FunBit)

Tredje brukertest var en ekstern test med bedriften FunBit. Under denne brukertesten ble det gitt svært positive tilbakemeldinger rundt den generelle tilnærmingen til utvikling av applikasjonen. Gruppen fikk også verdifull tilbakemelding på viktig funksjonalitet ved videre utvikling av systemet. En tilbakemelding var at bedrifter burde ha sin egen lukkede instans av applikasjonen for å ikke dele hemmelig bedriftsdata med konkurrenter (her Link Utvikling). En annen tilbakemelding var at fokus burde ligge på å støtte anbudskonkurranser for andre industrier enn IT-industrien, gjerne industrier som ofte har mindre teknisk kompetanse eller svarer på flere anbud (olje og bygg for eksempel).

Fjerde brukertest (Rainfall)

Fjerde og siste brukertest var med bedriften Rainfall. Her fikk to av de ansatte som jobber med utforming av anbud prøve systemet. Det ble gitt tilbakemelding om at det mest interessante i systemet per nå er muligheten for å få en oppsummering av anbudskonkurransen og avtaleteksten. For testerne er det svært interessant å kunne forkaste anbud man ikke kan svare på raskt, og de foreslo et "trafikklys"-system der bedriften kan definere hva som gjør det uaktuelt å gå videre med et anbud, og få et rødt, gult eller grønt lys etter hvor mange brudd konkurransen har på de etablerte retningslinjene til bedriften. De ønsket også å se hva slags dokumentasjon som forventes levert i anbudet, som for eksempel sikkerhetsklarering. Ved besvarelse av spørreundersøkelsen har testerne fra Rainfall kommet med ekstra kommentarer:

> *"Mye bra i applikasjonen. Vil nok trenge en del mer arbeid før den gir stort utbytte, men bare ved å gi oss en rask pekepinn på om vi skal svare opp på forespørselen vil den være nyttig."*
>
> *"Interessant konsept. Noe arbeid gjenstår før vi ville ha tatt det i bruk. Med noe forbedringer i oppsummeringen og kravspesifikasjon ser vi at vi kunne spart noe tid på å ta det i bruk."*

Resultatet av brukertestene

Etter brukertestene fylte testerne ut skjemaene beskrevet i kapittel 6.1.1.

Fra resultatene fra SUS-skjemaet ser vi at de fleste testere jevnt over hadde en grei brukeropplevelse med systemet (Tabell 6-4). Gjennomsnittlig poengsum har forbedret seg noe fra første til siste brukertest, men dette kan forklares med personlige preferanser og forventninger, da klientsiden ikke har blitt endret mye etter første brukertest.

Testerne fra Rainfall har lang erfaring med å utforme anbud i forhold til både FunBit og Link Utvikling. Denne brukertesten er også den eneste som er utført med en anbudskonkurranse som testerne er kjent med fra før. Eksterne testere, spesielt testere fra Rainfall mener gjennomgående at kvalitet på generert utdata er verre enn interne testere, særlig når det kommer til sammendrag av konkurranse, valgte kandidater og svar på kravspesifikasjon (Tabell 6-6).

Felles for alle testerne er at de ikke har hørt om en lignende løsning i markedet, og at de er interessert i å se applikasjonen utviklet og forbedret videre.

Tabell 6-4 - Resultat av SUS-skjema etter brukertester.

Spørsmål	**Test**						
	Internt			**FunBit**		**Rainfall**	
I think that I would like to use this system frequently	4	4	4	5	5	4	3
I found the system unnecessarily complex	1	1	3	1	2	1	2
I thought the system was easy to use	5	4	3	5	5	4	4
I think that I would need the support of a technical person to be able to use this system	1	1	1	1	2	1	1
I found the various functions in this system were well integrated	5	4	4	5	4	5	4
I thought there was too much inconsistency in this system	1	2	3	1	3	2	2
I would imagine that most people would learn to use this system very quickly	4	4	3	5	4	4	4
I found the system very cumbersome to use	1	2	2	1	4	1	1
I felt very confident using the system	5	4	3	5	4	4	4
I needed to learn a lot of things before I could get going with this system	3	2	4	1	2	2	1
Poengsum	90	80	60	100	72,5	85	80
Gjennomsnitt poengsum	76,7			86,25		82,5	

Tabell 6-5 - Svar på spørsmål om brukbarhet av anbudsgenerering

Spørsmål	**Test**						
	Internt			**FunBit**		**Rainfall**	
I am happy with the generated output	4	4	4	4	4	3	3
I don't think the generated output from the system is usable	1	2	1	1	2	2	2
I feel that this system saves me time	5	5	5	5	5	4	3
I would rather do this process manually	1	1	1	1	1	1	3
I think the system works as expected	4	4	4	4	4	4	3
I experienced many problems while using the system	1	2	2	1	3	2	2
Poengsum	55	50	52,5	55	47,5	45	35
Gjennomsnitt poengsum	52,5			51,25		40	

Tabell 6-6 - Svar på spørsmål om kvalitet på generert tekst

Spørsmål	**Test**				
	Internt	**FunBit**		**Rainfall**	
Kvalitet på sammendrag av avtaletekst (omfang av avtale, viktige punkter og lignende)	4	5	4	4	3
Kvalitet på introduksjon av bedriften (om bedriften og relevante tidligere prosjekter)	5	5	4	4	3
Kvalitet på sammendrag av konkurranse (hva skal produseres/leveres)	5	5	4	3	2
Kvalitet på valgte kandidater og begrunnelse av disse valgene (kompetansematrise)	4	5	3	3	3
Kvalitet på svar på kravspesifikasjon	3	5	4	3	2
Hvordan vil du beskrive personligheten til den genererte teksten?	Riktig	Riktig	For profesjonell - teksten føles overdrevent profesjonell og distansert	Riktig	Annet - Litt lite grunnlag til å avgjøre. Men selve språket i det jeg leste var OK.

6.2.2 Evaluering av utdata

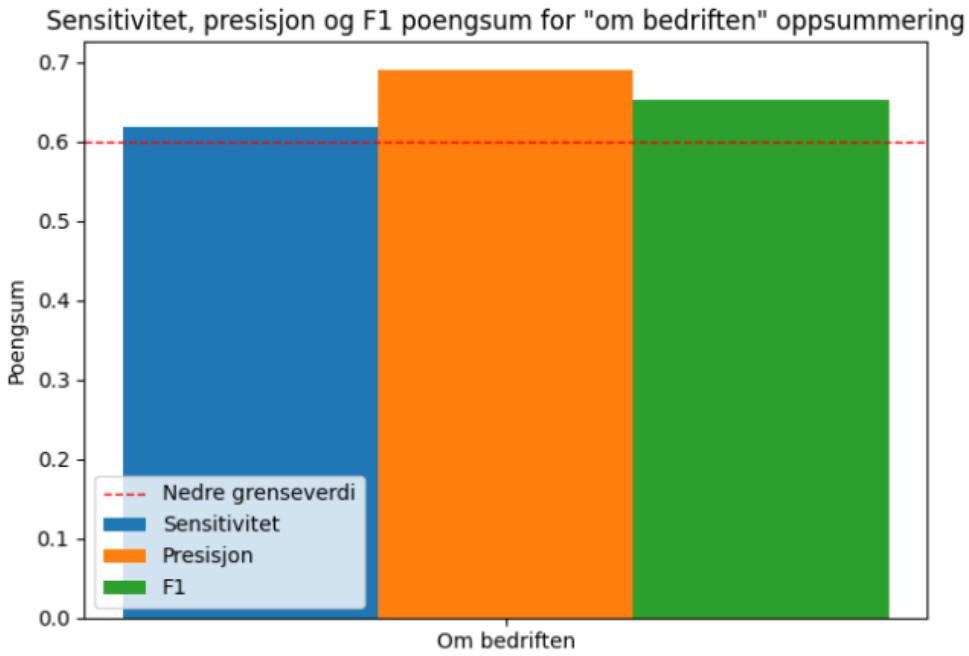
Det er utført testing med bruk av BERTScore på deler av applikasjonen, hvor slik testing er hensiktsmessig. Med hensiktsmessig menes der utdata som genereres er noenlunde forutsigbar, som for eksempel informasjon om bedriften. Andre genererte utdata vil være vanskelig å teste med BERTScore, da det ikke finnes noen referansesvar. Ved informasjon om bedriften testes informasjon slik den er generert for et anbud (kandidattekst), mot informasjonen som er hentet fra et tidligere anbud utformet av bedriften (referansetekst), siden denne informasjonen burde omhandle omtrent det samme. Ved leveranse av applikasjonen har informasjon om bedriften en sensitivitet på ~62%, presisjon på ~69%, og en F1 score på ~65%. Idf-vekting er ikke brukt da det er utfordrende å definere et datasett ut over kun referanseteksten som inneholder viktige ord for en spesifikk bedrift. Resultatene er vist i Figur 6-2.

Referansetekst (fra tidligere anbud)

> *"Link Utvikling AS er et prisbelønnet utviklerselskap som tilbyr app- og webløsninger, kreativt design og forretningsutvikling skreddersydd ditt behov. Vi kan ta hånd om alt fra planlegging og strategi, design og utvikling til drift. Link jobber i et tverrfaglig knutepunkt mellom innovasjon, kreativ prosjektledelse og teknologi. Teamet er satt sammen av mastergradskompetanse innen programvareutvikling, innovasjon og forretningsutvikling, og 15 års erfaring med brukerdrevet programvareutvikling. Selskapet har ved flere anledninger blitt priset for sine konsepter hvor vi blant annet har vunnet Innovasjonspriser og mottatt støtte fra Innovasjon Norge og Norsk Forskningsråd."*

Kandidattekst (GPT-3.5 Turbo)

> *"Link Utvikling ble grunnlagt i 2018 og er et anerkjent techbyrå spesialisert innen konsept- og programvareutvikling. Selskapet leverer tjenester som app- og webløsninger, kreativt design, forretningsutvikling, design og utvikling, konseptutvikling og programvareutvikling. Link Utvikling har et sterkt fokus på brukervennlighet gjennom grundig UX/UI-design. Deres lidenskap for arbeidet de utfører er tydelig, og dette vil du raskt merke etter første møte med gjengen i Link Utvikling."*



Figur 6-2 - Sensitivitet, Presisjon og F1 score for "om bedriften" oppsummering med BERTScore. Sensitivitet 0,62, presisjon 0,69, F1 0,65.

7 DISKUSJON

7.1 Prosjektgjennomføring

Problemstillingen vil nå gjennomgås for å se hvordan den utviklede løsningen relaterer til den gitte problemstillingen:

> *"Hvordan kan et system basert på kunstig intelligens utvikles for å effektivisere prosessen med anbudsskriving og kompetansesøk?"*

I løpet av prosjektperioden er det blitt utviklet en klient og tjener for generering av utkast til anbud. Løsningen inneholder et brukersystem med flere nivåer av brukere, mulighet for å legge til informasjon om bedrift og ansatte, og mulighet for å laste opp anbudskonkurranser og generere utkast til anbud for disse. Siden fokuset i prosjektet har vært på tjenersiden og utvikling av et API, har klientsiden blitt holdt minimal med lite fokus på design. Dette fordi Link Utvikling innehar stor kompetanse om design og utvikling av gode, brukervennlige løsninger, og de ønsket heller en mest mulig ferdig utviklet tjener.

Tabell 7-1 inneholder krav slik de er beskrevet i kapittel 2.1.2, og hvordan disse kravene er oppfylt i den endelige løsningen.

Tabell 7-1 - Krav til applikasjonen og hvordan disse er oppfylt

Krav	**Hvordan kravet er oppfylt**
Kunden kan logge seg inn og få tilgang til tjenesten.	Brukerhåndtering og tilgangsstyring er implementert ved hjelp av Firebase Authentication.
Kunder skal kunne lagre egne dokumenter i systemet, som for eksempel CV-er, tidligere anbud og informasjon om bedriften.	Kunder kan laste opp dokumenter i systemet som blir prosessert og lagret i MongoDB Atlas-databasen.
Det skal være mulig å laste opp filer fra en anbudskonkurranse i systemet.	Firebase Cloud Storage blir brukt til lagring av originale filer, mens MongoDB Atlas blir brukt for å lagre prosesserte filer for videre bruk i systemet.
Utkast til anbud skal kunne genereres med ett klikk.	Det er satt opp en knapp på klientside som, når trykket på, sender et API-kall til tjenersiden for å generere et anbud ut fra opplastede filer og valgte avtalemaler.
Utkast skal genereres ved hjelp av KI.	OpenAI-modellene GPT-3.5 Turbo og GPT-4 Turbo sammen med embedding-modellen Text Embedding 3 Small blir brukt for å generere utkast til anbud.
Løsningen skal kunne sette opp team med nødvendig kompetanse og lage en kompetansematrise, selv om ingen har nøyaktig den kompetansen som etterspørres.	Vektorsøk blir benyttet for å finne de ansatte med nærmest konkurranse, selv om ingen ansatte har nøyaktig riktig kompetanse.

Utkastet skal inneholde informasjon om bedriften og relevante tidligere prosjekter.	Blir generert ut fra innlagt informasjon om bedriften.
Utkastet skal inneholde informasjon om den foreslåtte løsningen der det er aktuelt.	For konkurranser som bruker avtalemalen SSA-T (Utvikling) genereres et forslag til løsning ut fra oppsummering av konkurransen og funnet kompetanse.
Utkastet skal inneholde svar på kravspesifikasjon der det er aktuelt.	For konkurranser som bruker avtalemalen SSA-T genereres svar på kravspesifikasjon ut fra oppsummering av konkurransen og foreslått løsning.

Alle krav stilt av oppdragsgiver i forkant av prosjektet er altså oppfylt. Det har imidlertid vært en del utfordringer og lærdommer tilknyttet utvikling av applikasjonen.

7.1.1 Brukertester

Gjennom de interne brukertestene med Link Utvikling kom det frem at de har et ønske om mer av de tekstlige beskrivelsene i et anbud (introduksjon av bedriften, relevante tidligere prosjekter og løsningsbeskrivelse). I tillegg kom det frem at de genererte svarene på kravspesifikasjon ikke nødvendigvis lagde en sammenhengende løsning som var mulig å utvikle. Ut fra dette har gruppen endret fokus mot tekstlige beskrivelser i anbudet, samt oppsummeringer av avtaletekst. Etter dette ble kravbesvarelsen forbedret ved å ta i bruk den tekstlige løsningsbeskrivelsen, noe som ga tydeligere rammer for svar på krav.

De siste to testene, med eksterne brukertestere fra FunBit og Rainfall, fungerte som sluttevaluering av produktet, der det utviklede produktet ble testet og evaluert, og mangler og ønsker for produktet ble kartlagt for eventuelt videre arbeid. En interessant observasjon fra sluttevalueringen er at testerne fra Rainfall, de som har mest erfaring med å utforme anbud, også er de som er minst positive til løsningen. Ettersom de har mer erfaring med anbud, ser de lettere feil og mangler i systemet. Dette viser at produktet har en lang vei å gå før det kan være et fullverdig hjelpemiddel for de som har lang erfaring med anbud.

En tilbakemelding som gikk igjen i alle brukertestene, er et ønske om å raskt kunne forkaste anbudskonkurranser som ikke passer til bedriften. Både Link Utvikling og Rainfall ønsker et system der man har "røde flagg" som vil varsle brukeren dersom anbudskonkurransen inneholder uakseptable vilkår og betingelser, eller dersom bedriften ikke innehar kompetanse til å levere på anbudet. Ettersom dette var et ønske som gikk igjen på tvers av alle brukertestene, burde dette være et stort fokusområde videre i utviklingen av applikasjonen.

I tillegg var svar på kravspesifikasjon jevnt over det testerne var minst overbevist av. Denne funksjonaliteten burde forbedres betraktelig, eller så burde løsningen skifte fokus vekk fra svar på utviklings- og tilpasningsavtalen SSA-T, og heller fokusere på å generere svar på bistandsavtalen SSA-B i første omgang.

7.1.2 Tilgang på data og kompetanse

Et stort problem har vært mangel på data. Gruppen hadde under store deler av utviklingen ikke tilgang på noen anbud å bruke som referanse. Medio februar fikk gruppen tilgang til et enkelt vinnende anbud å bruke som referanse, tidlig i april fikk gruppen også tilgang til et anbud levert av Link Utvikling. Totalt hadde gruppen altså tilgang til to anbud under utviklingen av prosjektet, der kun ett av dem var et vinnende anbud. Da oppdragsgiver heller ikke har lang erfaring med utforming av anbud, var det utfordrende å få klarhet i hva som burde være med i et anbud, og hvordan de forskjellige delene av anbudet best burde utformes. Det har dermed gått en del ressurser til å undersøke utforming av anbudskonkurranser og anbud. Dersom gruppen hadde hatt tilgang til mer data i form av ferdige anbud med de aktuelle konsulentenes CV-er, ville dette muliggjort testing av generert utdata opp mot det faktiske anbudet. Dersom gruppen hadde hatt jevnlig tilgang til en person med kompetanse innen utforming av anbud, ville dette gjort utviklingen av applikasjonen lettere da kravene til generering av anbud ville vært tydeligere og spørsmål rundt kvalitet og innhold kunne blitt avklart med denne personen.

7.1.3 Valg av teknologi

I starten av prosjektet var planen å ta i bruk en Firebase Cloud Firestore dokumentdatabase for lagring av data. Dette viste seg å ikke være ideelt da det var et behov for vektorsøk. Det ble derfor brukt en del tid på å undersøke alternative løsninger og flytte over databasen til den endelige løsningen, MongoDB Atlas. Denne endringen førte til en del dobbeltarbeid i starten av prosjektet, men valget om å bytte database har gjort det svært mye lettere å finne relevante dokumenter og kompetanse via vektorsøk med forhåndsfiltrering. Gruppen burde ha brukt mer tid på å undersøke databaser før oppstart av prosjektet.

Rammeverket Flask som ble brukt på tjenersiden er et minimalt rammeverk som gir veldig lite funksjonalitet, egnet for å lage et helt enkelt API. Dette gjorde at vi måtte bruke andre pakker og egendefinerte funksjoner for validering og sikkerhet. Et annet rammeverk som kunne blitt brukt er Django, og det gir det meste man trenger av funksjonalitet for å lage API-er, med innebygget validering, OpenAPI støtte, serialisering og sikring av endepunktene. Bruk av Django ville spart tid grunnet mer innebygget funksjonalitet.

Rammeverket LangChain er et ungt rammeverk i rask utvikling, noe som har vært merkbart under utvikling av prosjektet. Selv om LangChain tilbyr svært mye funksjonalitet som gjør utviklingsprosessen av språkmodell-applikasjoner mye lettere, er det en risiko å være avhengig av et rammeverk som når som helst kan gjøre omfattende endringer. Dette er imidlertid ikke bare et problem med LangChain, men også med hele det kommersielle språkmodell-landskapet. Under prosjektets tidsperiode har flerfoldige nye modeller blitt lansert, og det har kommet oppdateringer til OpenAI-modellene. Fordelen med LangChain er at rammeverket er leverandør-agnostisk, som betyr at man når som helst kan bytte til en annen leverandørs språkmodeller uten å skrive om hele kodebasen. I et raskt utviklende maskinlæringslandskap anså gruppen dette som en stor nok fordel til å veie tyngre enn risikoene ved å bruke LangChain, da det er veldig lett å bytte til en eventuell ny og bedre modell.

7.1.4 Store språkmodeller

Å jobbe med store språkmodeller fører med seg noen utfordringer. Som nevnt tidligere er dette et ungt landskap i rask utvikling, der nye og bedre modeller stadig blir sluppet. Dette fører til at det finnes få universelle retningslinjer eller "best practices" rundt utvikling av applikasjoner drevet av store språkmodeller, og mye av informasjonen som finnes er utdatert. Det er også mangel på robuste testsystemer for utdata fra generativ KI. Mange metoder for testing baserer seg på at man har et fasitsvar, noe som ikke er tilfelle i dette prosjektet. Gruppemedlemmene har etter beste evne jobbet for å forbedre prompter ut fra eksempel-anbud, faktasjekking opp mot anbudskonkurranse- og bedriftsdokumenter og retningslinjer for prompting fra OpenAI. Prosjektet kan ha stor nytte av å implementere mer robust testing av utdata etter hvert som løsninger for dette blir tilgjengelig.

Under prosjektets løp har gruppen lært mye om hvordan man skal skrive prompter og jobbe med store språkmodeller. Dette har vært en prosess med lesing og testing av ulike metoder for prompting, hvor gruppen har erfart at resultatene kan variere en del. Gruppen har også erfart at GPT-3.5 Turbo kan brukes til mye mer enn man opprinnelig hadde tenkt, som er ønskelig siden den er mye billigere og raskere enn GPT-4 Turbo.

Siden applikasjonen lar brukere laste opp egne dokumenter, og lar brukere fylle inn felter i nettportalen som brukes videre i prompter, er det en fare for prompt-injiserings-angrep. Prompt-injisering er en angrepsstrategi der en bruker av systemet kan få modellen til å overstyre instruksjonene sine ved å legge inn en prompt som ber modellen ignorere tidligere instruksjoner, og dermed leverer modellen et uønsket svar. Per nå er dette ikke vurdert som en stor risiko da modellen kun har tilgang på interne bedriftsdata og konkurransedokumenter til enhver tid. Dersom applikasjonen skal begynne å ta hensyn til eksterne data i generering av anbud, som for eksempel andre bedrifters data, må dette håndteres.

7.1.5 Refleksjon rundt valgte løsninger

I starten av utviklingen ble det diskutert at løsningen burde kunne generere et anbud med minimalt med interaksjon fra brukeren. I ettertid viste det seg at mer interaksjon fra brukeren kunne vært gunstig på flere punkter.

Dersom man hadde lagt inn funksjonalitet for å generere anbudet en seksjon av gangen, ville det vært lettere å la brukeren forbedre og finjustere hver del. Man kunne implementert en chat-funksjon der brukeren kan gi instruksjoner til språkmodellen for å forbedre seksjonen.

Det har vært vanskeligheter med å finne riktig fil og dokumentbit ved hjelp av vektorsøk. Slik det er nå, blir all teksten i en dokumentbit på 2000 tokener embeddet i sin helhet. Derfor kan embeddingen bli veldig lite spesifikk, da teksten kan handle om mange ulike ting. Ofte førte dette til at feil fil ble gitt som resultat, siden ingen filer hadde stor likhet. For å løse dette, kunne brukeren under opplasting av filer fått mulighet til å markere hvilken type fil det er, for eksempel avtalemal, kravspesifikasjon eller lignende. Dette ville gjort det lettere å finne riktig fil til hver seksjon av anbudsgenereringen. Eventuelt kunne man justert hvordan dokumenter blir delt opp i biter, og embeddet. Man kunne i stedet brukt språkmodellen til å finne nøkkelord av et dokument, og bruke nøkkelordene til å lage en embedding med, slik at vektoren i dokumentbitene ville blitt mer konkret, og forhåpentligvis ført til mer nøyaktige vektorsøk.

Om gruppen hadde hatt tilgang på mer data i form av ferdige anbud, kunne man finjustert modellen for å generere mer "anbud-lignende" svar. Dette var ikke mulig å få til grunnet mangel på data, men kunne forbedret resultatene betraktelig.

I tillegg kunne det vært gunstig å begrense omfanget av prosjektet ytterligere. Løsningen støtter SSA-B, SSA-B enkel og SSA-T, men SSA-T viste seg å være komplisert å generere anbud for. Det finnes mange mulige løsninger på en SSA-T-konkurranse, det blir vanskelig å finne den beste løsningen for bedriften og å generere en sammenhengende, fungerende løsning ut fra dette. Det er mange ulike kriterier som må tas hensyn til gjennom hele prosessen. SSA-B er lettere da det kun omhandler valg av riktig kompetanse ut fra veldig spesifikke krav. Dersom omfanget hadde blitt redusert, kunne kvaliteten på den produserte løsningen blitt bedre med mer funksjonalitet spesifikt for SSA-B, som en prisoversikt, bedre filtrering på valg av konsulenter, og hvilke ekstra dokumenter som forventes levert for en konkurranse.

7.1.6 Risikoutslag

I starten av prosjektet ble det gjort en risikovurdering (Vedlegg 3, Tabell 2-1). Noen av risikoene har inntruffet i prosjektets løp.

Risikoen for endringer i API har inntruffet, hovedsakelig grunnet endringer i rammeverket LangChain. Dette har ført til at deler av løsningen måtte endres underveis, men det største problemet har vært å finne oppdatert informasjon og dokumentasjon etter oppdateringer i rammeverket. Blant annet har LangChain nylig gått over til å bruke LCEL (LangChain Expression Language), som endrer hvordan kjeder blir satt opp. Ikke all dokumentasjon reflekterer dette ennå, og mye informasjon på nettet bruker den gamle metoden for å sette opp kjeder.

Videre var feilaktig tidsestimat vurdert til å ha middels høyt risikoprodukt. Under gjennomføring av prosjektet har gruppen estimert feil på en del oppgaver, uten at dette har ført til videre problemer. Risikoen hadde dermed lavere konsekvens enn antatt.

Lite sykdom med betydning for gjennomføringen har inntruffet i løpet av prosjektet, men et gruppemedlem var fraværende i siste del av prosjektet i forbindelse med militærtjeneste. Dette har ført til økt belastning for de resterende gruppemedlemmene i fraværsperioden, og for det gjeldende gruppemedlemmet med hensyn på å sette seg inn i endringer gjort under fraværet.

7.2 Etiske vurderinger

7.2.1 Kvalitetssikring

Ettersom store språkmodeller har en risiko for hallusinering, altså produksjon av uriktig tekst, må brukeren være kritisk til hva som blir produsert av modellen, og se over at det genererte anbudet er korrekt. I løpet av utviklingen av applikasjonen er det opplevd at språkmodellen tidvis hallusinerer, selv om tiltak for å minimere dette er tatt i bruk. Det er ingen måte å garantere at brukeren leser grundig gjennom det genererte anbudet, men nettportalen viser brukeren en advarsel om at generert utkast til anbud kan inneholde feilaktig informasjon, og at brukeren selv tar ansvar for å kontrollere informasjonen før eventuell leveranse av anbud.

Applikasjonen har også funksjonalitet for å lage oppsummeringer av konkurranser og tilhørende bestemmelser. En slik oppsummering vil spare brukeren for mye tid og gi et raskt overblikk av hva konkurransen handler om. Det er derimot også en risiko for at oppsummeringen utelater viktig informasjon om en konkurranse, som ville vært viktig for brukeren å være klar over før et anbud blir levert. I verste fall kan en bruker ende opp med å levere anbud på en konkurranse de ikke har full forståelse for. Også her vil det bli vist en advarsel til brukeren om at de selv er ansvarlig for å lese gjennom den fullstendige anbudskonkurransen før en eventuell leveranse.

Ettersom eier av Anbudsassistenten ikke kan garantere et feilfritt utkast til anbud, må en ansvarserklæring utformes der kunden går med på å selv ta ansvar for feilinformasjon i genererte oppsummeringer og anbud, og kunden plikter seg til å lese nøye gjennom anbud før leveranse.

7.2.2 Oppbevaring av data

Ved å bruke API-et til OpenAI blir disse dataene sendt til OpenAI sine tjenere. Ifølge OpenAI skal ikke disse dataene bli lagret eller brukt på noen måte. Likevel sendes dataene til utlandet ved kall til API-et, og er da underlagt andre regler for sikkerhet og databehandling. Etter tredje brukertest fikk gruppen tilbakemelding på at de fleste bedrifter ønsker å lagre data i Norge, slik at man er sikker på at dataene er trygge og følger GDPR, særlig når det kommer til data om ansatte. For å gjøre dette kan man bruke Azure AI Studio, som er en løsning for å bruke blant annet OpenAI API-et, der data blir lagret sikkert på en egen tjener. Ideelt sett bør også bedriften vedlikeholde sin egen instans av applikasjonen, hvor kun dataene til bedriften blir lagret. Da vil all data bli lagret lokalt, uten å forlate landet.

7.2.3 Bias i utdata fra store språkmodeller

Store språkmodeller er trent på enorme mengder tekstdata. Disse dataene er valgt ut av en gruppe personer, noe som forårsaker seleksjonsbias. Dataene inneholder sosiale biaser, som manifesterer seg i utdata fra modellen, blant annet når det kommer til kjønn, alder, legning og etnisitet (Navigli, Conia og Ross, 2023). Det er kritisk at valg av konsulenter til et anbud ikke er utsatt for sosiale biaser for at systemet i det hele tatt skal være brukbart. Mulighetene for å minimere sosiale biaser er begrenset dersom man ikke trener en egen språkmodell på nøye utvalgt og renset treningsdata. Ettersom applikasjonen bruker kommersielt tilgjengelige språkmodeller, vil en mulig fremgangsmåte være å rense inndata til modellen for all personalia som navn, alder og kjønn, men det er viktig å være klar over at en total eliminering av sosiale biaser er umulig grunnet biasen i dataene modellen er trent på. I denne applikasjonen finnes konsulenter ved hjelp av vektorsøk i databasen. Embeddinger for hver enkelt konsulent er renset for personalia, slik at navn, alder, kjønn og lignende ikke skal ha innvirkning på valg av konsulenter. All data for konsulenten blir derimot sendt til språkmodellen, inkludert personalia, men her er konsulentene allerede valgt. Det bør bli gjort videre vurderinger om dette er noe som har store innvirkninger på utdata og om det vil være gunstig å rense inndata til modellen også.

8 KONKLUSJON OG VIDERE ARBEID

Dette bachelorprosjektet hadde som mål å utvikle en løsning for å generere utkast til anbud som svar på anbudskonkurranser ved hjelp av kunstig intelligens. Problemstillingen som ble satt for oppgaven var:

> *"Hvordan kan et system basert på kunstig intelligens utvikles for å effektivisere prosessen med anbudsskriving og kompetansesøk?"*

Gjennom litteraturstudium ble det klart at lignende løsninger har blitt utviklet og at det derfor skal være mulig å lage et slikt system. Det har blitt undersøkt forskjellige kommersielt tilgjengelige språkmodeller og rammeverk for utvikling av prosjektet. Ved å benytte OpenAI sine GPT og embedding-modeller sammen med rammeverket LangChain og vektordatabasen MongoDB Atlas kan konkurransedokumenter, konsulentdata og bedriftsdata prosesseres, bli embeddet og lagres i vektordatabasen. Utkast til anbud kan bli generert i deler, med kjeder som tar inn nødvendig informasjon fra konkurransedokumenter, konsulenter og bedriften og sender forespørsler til OpenAI API-et. De resulterende delene kan settes sammen til et ferdig anbud og leses i nettportalen.

Målene som skulle oppnås var:

1. *Generere oppsummeringer av anbudskonkurranser ved å laste opp vedlegg fra konkurransen.*

Filer kan lastes opp i nettportalen gjennom brukergrensesnittet. Disse filene blir så prosessert, delt opp i biter og lagret i en database. To forskjellige oppsummeringer blir generert og returnert til bruker, et sammendrag av konkurransens avtaletekst og en oppsummering av hva som skal leveres eller produseres i konkurransen. Sammendrag av avtaletekst blir generert ut fra avtaletekst-dokumentet som følger med alle SSA-konkurranser. Oppsummering av hva som skal leveres eller produseres genereres ved hjelp av K-Means gruppering, en metode som sparer tid og penger ved å velge et representativt utvalg tekstbiter å oppsummere, i stedet for å oppsummere alle dokumentene i sin helhet.

2. *Skrive utkast til anbud, som kan oppdateres manuelt ved behov.*

Utkast til anbud blir generert ut fra standardavtalen(e) som er brukt for konkurransen, da forskjellige standardavtaler har forskjellige krav til anbud. Utkast til anbud genereres i deler som oppsummering, bedriftsintroduksjon, kompetanse og svar på kravspesifikasjon. Resultatet fra de ulike delene settes sammen og returneres som et helhetlig utkast og vises i brukergrensesnittet på nettportalen.

Per nå er det ingen måte å endre utkastet direkte i nettportalen, for å endre utkastet må teksten kopieres inn i et tekstredigeringsprogram. Beslutningen om å legge fokus på andre områder av løsningen ble gjort i samråd med veileder hos Link Utvikling, det ble enighet om at forbedring av utdata skulle bli prioritert.

3. *Finne riktig kompetanse basert på CV-er som er lastet opp.*

Nettportalen har en side der ansatte kan legges inn i systemet. Informasjon om ansatte kan legges inn manuelt, ved å laste opp CV, eller en kombinasjon av begge. Denne informasjonen

blir lagret i databasen. Det er mulig å benytte vektorsøk for å finne den mest relevante kompetansen for en konkurranse, dette gjøres ut fra et sammendrag av kravspesifikasjonen til konkurransen dersom det gjelder en utviklingsavtale, eller fra kravene for en bistandsavtale. Vektorsøk gjør det mulig å finne kompetanse som er relatert, men ikke nøyaktig lik. Hvis et krav er at den ansatte må ha kompetanse i React, kan en ansatt som har kompetanse i Vue.js bli returnert dersom ingen i bedriften har kompetanse i React.

Løsningen oppfyller de fleste målene som er satt for prosjektet. To av tre mål er fullstendig oppnådd, men løsningen mangler funksjonalitet for å eksportere eller redigere dokumentene i nettportalen.

8.1 Videre arbeid

Produktet som er utviklet under prosjektets løp er fungerende, men har begrenset funksjonalitet. Prosjektet har vist at utvikling av en anbudsassistent er mulig, og produktet støtter en begrenset mengde anbudskonkurranser. Det er en del punkter som kan forbedres ved videre arbeid på produktet, disse vil bli gjennomgått i dette kapittelet.

8.1.1 Utvikling av applikasjon

Oppsummering av vilkår og betingelser

Gjennomgående i alle brukertester var ønsket om en mer robust oppsummering av vilkår og betingelser for en anbudskonkurranse, spesifikt tilpasset en bedrift. Ønsket er å raskt kunne forkaste anbudskonkurranser som har uakseptable vilkår. Et "trafikklys"-system burde utvikles der bedrifter kan spesifisere hva som er uakseptabelt, og vilkår som burde sjekkes nøyere før en eventuell besvarelse.

Generering av anbud

Det genererte utkastet for anbud har stort forbedringspotensial. For å spisse utdata fra modellen enda mer, kan det være aktuelt å holde en samtale med modellen for å endre på spesifikke deler av anbudet. Denne funksjonaliteten vil være komplisert å implementere, da applikasjonen sender mange ulike prompter og samler mye data for å generere et anbud, som settes sammen til slutt. Det burde ikke være nødvendig å generere hele anbudet på nytt hver gang brukeren ønsker en endring, derfor må det finnes en måte å kun oppdatere den spesifikke delen av anbudet kommentaren gjelder. En mulig løsning er å gjøre anbudsgenerering til en prosess der en og en del av anbudet genereres og finjusteres om gangen. Det er også ønskelig å mate modellen med informasjon om tidligere vinnende anbud bedriften har levert.

Det kan utvikles en løsning for å laste ned generert anbud i et bestemt format, for eksempel som et Word-dokument eller en PDF-fil, ut fra dataene som ligger i databasen. Utvikling av slik funksjonalitet vil kreve en del ressurser da formatering må settes opp manuelt.

Innhenting av ekstern data

Mercell og Doffin har sine egne API-er hvor man kan hente informasjon om anbudskonkurranser og filer, og sette opp varslinger for interessante anbudskonkurranser. Dette kan integreres med applikasjonen for å finne relevante anbudskonkurranser for enhver bedrift, for å strømlinjeforme anbudsprosessen enda mer.

Å legge inn bedriftsdata kan ta en del tid dersom man har mye data som kan legges inn. For å gjøre denne prosessen raskere og enklere, kan man bruke OpenAI API-et med nettsøk for å hente informasjon fra bedriften sin nettside eller andre kilder.

Klient

Brukergrensesnittet til applikasjonen har vært lavt prioritert i løpet av utviklingsperioden, og trenger å forbedres med hensyn på brukervennlighet, universell utforming og estetikk. En del av brukervennligheten som kunne vært forbedret er ventetiden i det man genererer et anbud eller en oppsummering. Slik applikasjonen fungerer nå så sendes hele resultatet fra tjeneren når det er ferdig, men det er mulig å sende utdata så fort deler av det er ferdig, såkalt strømming av data. Dette vil ikke gjøre genereringen raskere, men det vil føles mer responsivt for brukeren.

8.1.2 Utvidet støtte

Applikasjonen støtter i første omgang standardavtalene SSA-B, SSA-B enkel og SSA-T, men det er flere andre avtaler som burde støttes. Før applikasjonen gjøres kommersielt tilgjengelig, kan det være fornuftig å støtte alle standardavtalene for IT-konkurranser.

Dersom applikasjonen skal lanseres, vil gjerne de største kundene være de som ikke innehar så mye teknisk kompetanse selv, eller bedrifter i industrier som leverer et større antall anbud. På sikt kan applikasjonen derfor utvides til å støtte anbudskonkurranser for blant annet olje- og byggeindustrien.

9 REFERANSER

Anthropic (2024a) *Claude API*. Tilgjengelig fra: https://www.anthropic.com/api (Hentet: 19 april 2024).

Anthropic (2024b) *Rate limits, Claude*. Tilgjengelig fra: https://docs.anthropic.com/claude/reference/rate-limits (Hentet: 19 april 2024).

Brooke, J. (1995) 'SUS: A quick and dirty usability scale', *Usability Eval. Ind.,* 189.

Dair.AI (2024a) Few shot prompting, *Few shot prompting*. Tilgjengelig fra: https://www.promptingguide.ai/techniques/fewshot (Hentet: 6 mars 2024).

Dair.AI (2024b) *Prompt Chaining, Prompt Chaining.* Tilgjengelig fra: https://www.promptingguide.ai/techniques/prompt_chaining (Hentet: 6 mars 2024).

Dair.AI (2024c) *Zero shot prompting, zeroshot*. Tilgjengelig fra: https://www.promptingguide.ai/techniques/zeroshot (Hentet: 6 mars 2024).

Direktoratet for forvaltning og økonomistyring (2024) *Hva er innovative anskaffelser? | Anskaffelser.no*. Tilgjengelig fra: https://anskaffelser.no/innovasjon/hva-er-innovative-anskaffelser (Hentet: 19 februar 2024).

Doffin, Database for offentlige anskaffelser (2024a) *SSA-B og SSA-B enkel (Bistandsavtalene) | Anskaffelser.no.* Tilgjengelig fra: https://anskaffelser.no/verktoy/maler/ssa-b-og-ssa-b-enkel-bistandsavtalene (Hentet: 6 mars 2024).

Doffin, Database for offentlige anskaffelser (2024b) *SSA-T (Utviklings- og tilpasningsavtalen) | Anskaffelser.no*. Tilgjengelig fra: https://anskaffelser.no/verktoy/maler/ssa-t-utviklings-og-tilpasningsavtalen (Hentet: 6 mars 2024).

Doffin, Database for offentlige anskaffelser (2024c) *Statens standardavtaler (SSA) | Anskaffelser.no*. Tilgjengelig fra: https://anskaffelser.no/avtaler-og-regelverk/statens-standardavtaler-ssa (Hentet: 6 februar 2024).

Erdem, K. (2021) *Understanding Positional Encoding in Transformers, Understanding Positional Encoding in Transformers - Blog by Kemal Erdem*. Tilgjengelig fra: https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers#positional-encoding-visualization (Hentet: 24 april 2024).

Falkner, A. et al. (2019) 'Identifying Requirements in Requests for Proposal: A Research Preview', E. Knauss and M. Goedicke (eds) *Requirements Engineering: Foundation for Software Quality*. Cham: Springer International Publishing (Lecture Notes in Computer Science), s. 176–182. doi: 10.1007/978-3-030-15538-4_13.

Firebase (2024a) *Control Access with Custom Claims and Security Rules | Firebase Authentication, Firebase*. Tilgjengelig fra: https://firebase.google.com/docs/auth/admin/custom-claims (Hentet: 25 mars 2024).

Firebase (2024b) *Verify ID Tokens | Firebase Authentication, Firebase*. Tilgjengelig fra: https://firebase.google.com/docs/auth/admin/verify-id-tokens (Hentet: 25 mars 2024).

Howard, J., Gugger, S. og Chintala, S. (2020) *Deep learning for coders with fastai and PyTorch: AI applications without a PhD*. First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly.

Hugsted, R. og Anderssen, H.B. (2023) 'anbud', *Store norske leksikon*. Tilgjengelig fra: https://snl.no/anbud (Hentet: 6 Februar 2024).

Kristiansen, N. og Vatnelid, T. (2024a) *DAT255 Course Project - Exploration of summarization techniques using machine learning*. Tilgjengelig fra: https://www.norakristiansen.no/dat255-summarization/ (Hentet: 24 april 2024).

Kristiansen, N. og Vatnelid, T. (2024b) *DAT255 Course Project - KMeans Clustering*. Tilgjengelig fra: https://www.norakristiansen.no/dat255-summarization/posts/kmeans-clustering/clustering.html (Hentet: 24 april 2024).

Krüger, K. og Anderssen, H.B. (2023a) 'anbudskonkurranse', *Store norske leksikon*. Tilgjengelig fra: https://snl.no/anbudskonkurranse (Hentet: 6 februar 2024).

Krüger, K. og Anderssen, H.B. (2023b) 'offentlig anskaffelse', *Store norske leksikon*. Tilgjengelig fra: https://snl.no/offentlig_anskaffelse (Hentet: 6 februar 2024).

Lakeworks (2024) 'Scrum (software development)', *Wikipedia*. Tilgjengelig fra: https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=1215646479 (Hentet: 28 mars 2024).

Link Utvikling AS (2024) *Prosjekter | App, Web & Design | Link Utvikling*. Tilgjengelig fra: https://www.linkutvikling.no/prosjekter (Hentet: 1 mars 2024).

Meta (2024) *Meta Llama 2, Meta Llama*. Tilgjengelig fra: https://llama.meta.com/llama2/ (Hentet: 19 april 2024).

Motahari-Nezhad, H.R. et al. (2016) 'RFPCog: Linguistic-Based Identification and Mapping of Service Requirements in Request for Proposals (RFPs) to IT Service Solutions', *2016 49th Hawaii International Conference on System Sciences (HICSS). 2016 49th Hawaii International Conference on System Sciences (HICSS),* s. 1691–1700. doi: 10.1109/HICSS.2016.213.

Navigli, R., Conia, S. og Ross, B. (2023) 'Biases in Large Language Models: Origins, Inventory, and Discussion', *Journal of Data and Information Quality*, 15(2), s. 10:1-10:21. doi: 10.1145/3597307.

Neelakantan, A. et al. (2022) *Introducing text and code embeddings*. Tilgjengelig fra: https://openai.com/blog/introducing-text-and-code-embeddings (Hentet: 4 mars 2024).

Nistala, P. et al. (2023) 'An industrial experience report on model-based, AI-enabled proposal development for an RFP/RFI', *Science of Computer Programming,* 233, s. 103058. doi: 10.1016/j.scico.2023.103058.

OpenAI (2022) *ChatGPT - Release Notes*. Tilgjengelig fra: https://help.openai.com/en/articles/6825453-chatgpt-release-notes (Hentet: 22 februar 2024).

OpenAI (2024a) *OpenAI | Assistants API*. Tilgjengelig fra: https://platform.openai.com (Hentet: 19 januar 2024).

OpenAI (2024b) *OpenAI | Embeddings*. Tilgjengelig fra: https://platform.openai.com/docs/guides/embeddings/embedding-models (Hentet: 3 mai 2024).

OpenAI (2024c) *OpenAI | Pricing*. Tilgjengelig fra: https://openai.com/pricing (Hentet: 17 januar 2024).

OpenAI (2024d) *OpenAI | Prompt engineering, Prompt enginerring*. Tilgjengelig fra: https://platform.openai.com/docs/guides/prompt-engineering (Hentet: 17 januar 2024).

OpenAI (2024e) *Openai Models*. Tilgjengelig fra: https://platform.openai.com/docs/models (Hentet: 8 mai 2024).

OpenAI (2024f) *OpenAI Platform*. Tilgjengelig fra: https://platform.openai.com (Hentet: 3 april 2024).

Papineni, K. et al. (2002) 'Bleu: a Method for Automatic Evaluation of Machine Translation', P. Isabelle, E. Charniak, and D. Lin (eds) *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. ACL 2002*, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, s. 311–318. doi: 10.3115/1073083.1073135.

Rehkopf, M. (2024) *What is a Kanban Board?,* Atlassian. Tilgjengelig fra: https://www.atlassian.com/agile/kanban/boards (Hentet: 28 mars 2024).

Schwaber, K. og Sutherland, J. (2020) *Scrum Guide | Scrum Guides*. Tilgjengelig fra: https://scrumguides.org/scrum-guide.html (Hentet: 28 mars 2024).

Scikit Learn (2024a) *Scikit learn - KMeans clustering, scikit-learn*. Tilgjengelig fra: https://scikit-learn.org/stable/modules/clustering.html (Hentet: 3 mai 2024).

Scikit Learn (2024b) *Selecting the number of clusters with silhouette analysis on KMeans clustering, scikit-learn.* Tilgjengelig fra: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html (Hentet: 6 mai 2024).

Tianyi (2024) *'Tiiiger/bert_score'*. Tilgjengelig fra: https://github.com/Tiiiger/bert_score (Hentet: 5 april 2024).

Tidemann, A. (2024) 'kunstig intelligens', *Store norske leksikon*. Tilgjengelig fra: https://snl.no/kunstig_intelligens (Hentet: 1 mai 2024).

Truong, T.H. et al. (2023) ‘Language models are not naysayers: an analysis of language models on negation benchmarks’, *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023),* Toronto, Canada: Association for Computational Linguistics, s. 101–114. doi: 10.18653/v1/2023.starsem-1.10.

Umbrel (2024) ‘getumbrel/llama-gpt’. *Umbrel*. Tilgjengelig fra: https://github.com/getumbrel/llama-gpt (Hentet: 14 februar 2024).

Vaswani, A. et al. (2017) ‘Attention is All you Need’. *arXiv preprint* arXiv.1706.03762.

Zhang, T. et al. (2020) ‘BERTScore: Evaluating Text Generation with BERT’. *arXiv preprint* arXiv:1904.09675v3.

Ziv, R. og Anadkat, S. (2024) *Getting Started with OpenAI Evals | OpenAI Cookbook*. Tilgjengelig fra: https://cookbook.openai.com/examples/evaluation/getting_started_with_openai_evals (Hentet: 29 april 2024).

10 VEDLEGG

Vedlegg 1 - Visjonsdokument

Vedlegg 2 - Kravdokument

Vedlegg 3 - Prosjekthåndbok

Vedlegg 4 - Systemdokumentasjon

Vedlegg 5 - Kildekode