Fakultet for Ingeniørvitenskap og Teknologi
Institutt for Datateknologi og Beregningsorientert Ingeniørfag

Sluttrapport Bachelor

Fosby, Martin Johan Holand
Furunes, Jakob
Selfors, Thomas

Bacheloroppgave i Datateknologi, 2025 - mai - 20

UiT Norges arktiske universitet

Tittel: Optimalisering av Feilhåndtering og Datafangst i Bodø Kommune: En Analyse av Digitalisering og IKT-Strategier	**Dato**: 20.05.2025
	Gradering: Åpen
Forfattere: Martin Johan Holand Fosby, Thomas Selfors, Jakob Furunes	**Antall sider**: 35
	Antall vedlegg: 1 (Zip-fil med kode)

Fakultet: IVT	**Institutt**: IDBI

Studieretning:
Datateknologi

Veileder:
Hans Richard Movik, Hans Olofsen

Oppdragsgiver:
Bodø Kommune

Oppdragsgivers kontaktperson:
Bjørn Inge Furunes

Sammendrag:
Sluttrapport med gjennomgang og diskusjon av system designet for å motta og behandle feilmeldinger ved hjelp av teknologier som NB-Whisper og retrieval-augmented generation.

Abstract:
Final report with explanation and discussion of system designed to gather and analyze error messages using technologies such as NB-Whisper and retrieval-augmented generation.

Forord

Takk til Bodø kommune og Bjørn Inge Furunes for å ha bidratt med oppgaven, og med støtte underveis i designet og utviklingen av produktet. Takk til veiledere Hans Richard Movik og Hans Olofsen for hjelp til utførelse og skriving av bacheloroppgaven.

Martin Johan Holand Fosby, Jakob Furunes, Thomas Selfors
Bodø, 20.05.2025

Sammendrag

På vegne av Bodø kommune har vi undersøkt problemstillingen rundt automatisk innmelding og behandling av feilmeldinger. Vi har designet et produkt som mottar feilmeldigner via skjema eller KI-transkribert tale-til-tekst ved innringing, og deretter benytter en retrieval-augmented generation-løsning til å analysere feilen og finne lignende feil i gamle hendelser. Løsningen leverer gjenkjennbar transkribering, men sliter noe med stedsnavn og navn på tjenester. Klassifiseringen av feilmeldinger er god, men avviker noen steder fra Bodø kommunes vurdering. Oppsummert er produktet bra og viser tydelig at teknologien kan bidra til forbedringer på dette området, men at det fortsatt er mangler som gjør at den ikke nødvendigvis bør brukes som eneste verktøyet i innsamling og prosessering av feilmeldinger. Produktet kan videreutvikles senere eller brukes som inspirasjon til liknende produkter.

Innhold

1 Introduksjon

1.1 Beskrivelse av oppdragsgivers behov

Digitaliserings- og IKT-kontoret til Bodø kommune forvalter IKT-systemene som brukes i kommunen. Dette inkluderer blant annet helsejournaler, trådløspunkter og skole-PC-er. Ved feil i disse systemene, skal brukeren melde inn feilen til brukerstøtte, som deretter er ansvarlig for å rette feilen. For å bestemme hvor alvorlig feilen er, blir feilen klassifisert med en prioritetsgrad fra 1-4, hvor 1 er en kritisk feil mens 4 er en feil med lav alvorlighetsgrad. Det er visse kriterier som setter føringer på hvordan man skal kategorisere en feil, for eksempel vil feil som berører helsetjenesten klassifiseres som kritiske ettersom de kan påvirke liv og helse.

Ved feil i systemene er det viktig å kategorisere feilen tidlig i feilrettelsesprosessen, slik at man kan delegere riktig personell til å fikse feilen og fastsette hvor lang tid det maksimalt skal ta før feilrettingen påbegynnes. Det er avtalefestet i en Service Level Agreement (SLA) mellom Bodø kommune og deres leverandører hvor lang tid leverandørene skal bruke på å begynne å rette feil, avhengig av hvor alvorlig feilen er. For eksempel skal reparatører begynne feilsøking og reparering innen 20 minutter hvis feilen er kritisk, mens det er 4 timer på en moderat feil. Dersom en feilmelding blir mistolket og feilen ikke får korrekt klassifisering, kan det i verste fall føre til at ressursene som blir brukt ikke er tilstrekkelige til å fikse feilen innen optimal tid, og at kritiske systemer ikke har for lang nedetid før det blir håndtert.

Bodø kommune har laget en kritikalitetsmatrise som fungerer som en veileder for klassifisering av feil. Matrisen beskriver systemene som brukes av kommunen og er delt opp i to deler, hvor den ene delen gjelder systemer som brukes av helse- og omsorgstjenesten, og den andre gjelder generelle systemer. Hvert system har en innvirkningsvurdering (impact) basert på hvor kritisk systemet er for drift, og en hastevurdering (urgency) som baser seg på hvor mange brukere som er påvirket av feilen. Sammen bestemmer de to vurderingene hvilken prioritering feilen skal ha. For eksempel har journaltjenesten Gerica en høy innvirkningsvurdering og medium eller høy hastevurdering, avhengig av om problemet har en workaround eller ikke, og til sammen gjør dette at feil på Gerica skal klassifiseres til kritisk eller høy prioritering.

Bodø kommune ønsker å forbedre klassifiseringsprosessen slik at samsvaret mellom feilen som er meldt inn og kriteriene for klassifisering er mer nøyaktige. Kort forklart betyr det at Bodø kommune mener at klassifiseringene som blir gjort av dagens system ikke er nøyaktige nok og kan gi feil bilde av feilen. Ukorrekte klassifiseringer kan føre til nedetid for viktige systemer, og påvirker den daglige driften av Bodø kommune. De ønsker også at det skal være lettere å finne tidligere hendelser som deler likheter med innmeldt feilmelding, slik at de kan bruke tidligere erfaringer og løsninger til å bidra med å rette feilen.

1.1.1 Behov for forbedring

Bodø kommune ønsker å forbedre klassifiseringsprosessen slik at samsvaret mellom feilen som er meldt inn og kriteriene for klassifisering er mer nøyaktige. Kort forklart betyr det at Bodø kommune mener at klassifiseringene som blir

gjort av dagens system ikke er nøyaktige nok og kan gi feil bilde av feilen. Feile klassifiseringer kan føre til nedetid for viktige systemer og påvirker den daglige driften av Bodø kommune. De ønsker også at det skal være lettere å finne tidligere hendelser som deler likheter med innmeldt feilmelding, slik at de kan bruke tidligere erfaringer og løsninger til å bidra til å rette feilen.

1.1.2 Forslag til feilforslag

I tillegg til automatisk klassifisering ønsker oppdragsgiveren et system som kan generere forslag til mulige feilhendelser. Dette skal bidra til å automatisere feilbehandlingsprosessen ytterligere og redusere kostnader. Målet er å hjelpe brukerstøtte med å raskere identifisere mulige årsaker til feilen, slik at utbedring kan igangsettes så tidlig som mulig. Oppdragsgiveren har et spesielt håp om at kunstig intelligens-baserte systemer, som for eksempel Retrieval Augmented Generation (RAG), kan spille en sentral rolle i denne sammenhengen.

1.1.3 Innkast av feilmeldinger

Oppdragsgiver ønsker at produktet skal benytte telefoni som et medium for innsendelse av feilmeldinger. I tillegg har vi valgt å inkludere muligheten for å sende inn feilmeldinger via et skriftlig skjema. Begge metodene benytter samme underliggende system for mottak og behandling av innkommende informasjon, noe som forenkler integrasjonen og gir fleksibilitet for brukerne.

1.1.4 Transkribering av feilmeldinger

Oppdragsgiveren ønsker at taleopptak fra telefonhenvendelser automatisk transkriberes til tekst. For dette formålet ønsker de å benytte NB-Whisper, en norsktilpasset variant av OpenAIs Whisper-modell. Ved å implementere et slikt system for transkribering, vil man kunne i teorien behandle feilmeldinger sendt via tale på lik linje med skriftlige henvendelser, noe som bidrar til en mer effektiv og enhetlig håndtering av innkommende meldinger.

1.2 Problemstilling og hypotese

Vi ønsker å utvikle et produkt som automatisk mottar og klassifiserer feilmeldinger, hvor klassifiseringen følger et bestemt regelverk slik at det gjøres konsekvente vurderinger av hvor kritisk feilene er, og at klassifiseringen er nøyaktig og korrekt i forhold til regelverket. Problemet med dagens løsning, hvor kundebehandlere mottar feilmeldinger fra telefon eller epost, er at det er avhengig av hver enkelt kundebehandlers vurdering av regelverket. Vårt system skal bestemme klassifisering direkte fra kriteriene bestemt av Bodø kommune, noe som vil føre til mer nøyaktige klassifiseringer. Dette er u tråd med målsettingen fra oppdragsgiver, som i oppgaveteksten (1) fremhever at en automatisert klassifisering med RAG metodologi vil "kunne øke kvaliteten på feilhåndteringen" og redusere tidsbruk.

Vår hypotese er at produktet vårt kan forbedre klassifiseringsprosessen, slik at klassifiseringen hver feilmelding får, er bestemt med bakgrunn i kriteriene fastsatt av Bodø kommune. Vi mener også at tiden det tar fra innmeldt feilmelding til ferdig klassifisering vil minske betraktelig. I sum vil dette bety at man vil få korrekte klassifiseringer på kortere tid enn med dagens system.

Til slutt mener vi at det vil være en kostnadsbesparelse for Bodø kommune, ikke bare i form av at det ikke blir brukt unødvendige ressurser til feilretting, men også at kostnadene til drift av produktet vil være lavere enn dagens kostnader. Vi vil gå inn på en mer detaljert oversikt over kostnader knyttet til produktet senere i rapporten.

1.3 Oversikt over rapportens innhold

Rapporten vil først gå gjennom den teoretiske bakgrunnen som ligger til grunn for teknologien som har blitt brukt, og som former valgene vi har gjort under utviklingen. Ett av hovedmomentene i utviklingen har vært bruk av KI til å behandle språk, enten fra tale til tekst eller til å tolke feilmeldinger, og vi vil kort gjøre rede for teorien som ligger bak KI-metodene vi har benyttet i produktet. Deretter vil vi drøfte og forklare valgene vi har gjort i forhold til bruk av teknologi og utviklingsmetode, blant annet forklare hvorfor vi har valgt å benytte RAG-teknologi til å behandle og prosessere feilmeldingene.

Vi vil se på resultatet av det ferdigstilte produktet og hvordan det utfører oppgavene sine i forhold til hypotesen vår. Denne delen av rapporten vil vise tidsbruk for behandling av innmeldt feilmelding, hvor treffsikker klassifisering som er utført av systemet er, og til slutt se på nøyaktigheten til transkribering av tale til tekst. I neste del av rapporten vil vi diskutere resultatene og deres betydning satt opp mot løsningen Bodø kommune bruker nå, og hvilken samfunnsmessig innvirkning produktet vårt vil ha. Rapporten avsluttes med en oppsummering av argumentene vi har lagt fram i rapporten og en konklusjon av hva vi mener produktet har oppnådd, og hvordan vi har løst problemstillingen ved å utvikle et produkt som leverer rask og konsekvent analyse av feilmeldinger.

2 Teori: drøfting av mulige teknologier og utviklingsmetoder

2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP), eller naturlig språkprosessering, er et fagfelt innen kunstig intelligens og datalingvistikk som fokuserer på å gjøre det mulig for datamaskiner å forstå, tolke og generere menneskelig språk. For vårt feilklassifiseringssystem er NLP-teknikker essensielle både for å forstå innholdet i selve feilmeldingene, samt klassifisere dem korrekt basert på kritikalitetsmatrisen. Teknologien har et bredt bruksområde, inkludert talegjenkjenning, maskinoversettelse, sentimentanalyse og informasjonsuthenting [1].

En nyere utvikling innen NLP er bruken av såkalte henteforsterkede generative modeller, ofte kalt Retrieval-Augmented Generation (RAG). Dette er en metoden vi vil anvende, og vil gå dypere i dens formål for vårt system senere i rapporten. Denne modellen kan forberede relevant kontekst fra en ekstern kunnskapsbase før den genererer tekst [2]. Denne teknikken er spesielt nyttig i oppgaver der det er behov for oppdatert eller domene-spesifikk informasjon, og der språkmodellen i seg selv ikke har tilstrekkelig kunnskapsgrunnlag.

Et annet viktig område innen NLP er automatisk talegjenkjenning, som innebærer å konvertere talte ord til tekst. Dette er en kompleks oppgave, da tale har varierende faktorer som dialekt, tempo og uttale, og krever sofistikert akustisk modellering og språklig kontekstforståelse [3]. Slike systemer benytter ofte dype nevrale nettverk trent på store mengder lyd-tekst-par for å lære sammenhengen mellom fonemer og ordsekvenser.

NLP er viktig for all analyse av tekst. I Bodø kommune kommer feilmeldinger som fritekst, så systemet er nødt å være i stand til å forstå det norske språk, med dets varianter av dialekter så naturlig som mulig.

2.2 Nevrale nettverk (NN)

Nevrale nettverk (Neural Networks, NN), spesielt kunstige nevrale nettverk, er samlinger av kunstige nevroner inspirert av hjernen til biologiske organismer. Nevronene er forbundet med hverandre gjennom vekter, som er justerbare parametere. Disse vektene kan anses som analoge med synapsene i biologiske nevrale nettverk. Aktiveringsfunksjoner anvendes i hvert nevron for å bestemme utgangssignalet basert på summen av de mottatte signalene. [4]

Fordelen med et slikt nettverk er evnen til å lære komplekse mønstre automatisk, men hovedutfordringen for vårt brukstilfelle er behovet for store menger merkede treningsdata (tidligere feilmeldinger) hvor man er sikre på at data er riktig klassifisert, i tillegg til at det krever mer regnekraft, kontra å anvende en forhåndstrent modell.

2.3 Transformerarkitektur

Transformer-arkitekturen, introdusert av Vaswani et al, er kjernen i moderne språkmodeller. [5]. Denne arkutekturen bruker *selvoppmerksomhetsmekanismer* (self-attention) for å analysere innbyrdes relasjoner mellom ord, uavhengig av

ordenes posisjon i teksten. Dette gjør at modellen er godt egnet til å forstå selve betydningen og konteksten i komplekse formuleringer, som ofte er realistiske forekomster, særlig i tekniske feilmeldinger for vårt formål.

Denne arkitekturen opererer med "tokens", eller språkenheter på norsk. Språkenheter kan være hele ord eller deler av ord. Når et ord deles opp i flere tokens, kalles det subword-tokenisering. Tokens kan dermed ses på som små byggesteiner av tekst som modellen forstår, og arbeider med.

Begrepet "token" ble brukt i NLP før Transformer, men Transformer-arkitekturen standardiserte bruken av tokens i moderne nevrale språkmodeller.

Tokens blir deretter konvertert til vektorer, som er numeriske representasjoner av tokens. Dette er nødvendig fordi nevrale nettverk håndterer numeriske data (vektorer og matriser) mye mer effektivt enn tekst. Vektorene gjør det mulig for modellen å forstå egenskaper og relasjoner mellom tokens på en måte som gjør det lettere å trekke meningsfulle sammenhenger mellom ord eller deler av ord.

En av de tidlige metodene for å lære vektorrepresentasjoner var Word2Vec [6]. I moderne arkitekturer, som Transformer-arkitekturen, benyttes egne trenbare "embedding-lag" for å lære slike representasjoner. Disse embeddingene trenes samtidig som resten av modellen, og de tilpasses oppgaven som modellen er trent på.

Transformer-arkitekturen utgjør grunnlaget for mange av dagens store språkmodeller (LLM-er). Innen transformerbaserte modeller finnes ulike varianter som benytter enten kun en encoder, kun en decoder, eller en kombinasjon av begge (encoder-decoder).

Encoder-modellen anvendes primært for å forstå og representere kontekstuell informasjon i inndata, mens decoder-modellen hovedsakelig benyttes til generering av tekst eller sekvenser. Encoder-decoder-arkitekturen kombinerer disse funksjonene ved først å kode inn inndata med encoderen, for deretter å generere utdata via decoderen.

En sentral fellesnevner i disse variantene er bruken av vektorbaserte representasjoner av tokens samt "self-attention"-mekanismen, som muliggjør modellens evne til å vurdere relasjoner mellom ulike deler av inndataene uavhengig av sekvenslengde.

2.4 Store språkmodeller

Store språkmodeller (eng. Large Language Models, LLMs) er statistiske maskinlæringssystemer som benytter dype nevrale nettverk for å modellere og generere naturlig språk. De er vanligvis basert på transformer-arkitekturen introdusert av Vaswani et al. (2017), som muliggjør effektiv håndtering av sekvensielle data gjennom selvoppmerksomhetsmekanismer. Dette skiller dem fra tidligere tilnærminger som rekurrente nevrale nettverk (RNNs), ved at transformere tillater parallell prosessering og langt mer effektiv trening på store datasett [7].

LLMs trenes på store korpus av tekstdata for å lære statistiske regulariteter i språk. Under treningen estimerer modellen sannsynligheten for neste ord gitt en tidligere kontekst, noe som gjør den i stand til å generere sammenhengende

og grammatisk korrekt tekst. Et viktig kjennetegn ved slike modeller er at de, på grunn av størrelsen og omfanget på treningsdataene, kan lære generaliserte representasjoner av språk, kunnskap og semantikk uten eksplisitt merking.

Bruksområdene for store språkmodeller er mangfoldige og inkluderer oppgaver som maskinoversettelse, tekstoppsummering, spørsmålsbesvarelse, informasjonsgjenfinning og dialogsystemer. De kan også benyttes som grunnlag for few-shot eller zero-shot læring, hvor de løser nye oppgaver med svært få eller ingen eksempler, ved hjelp av promptbasert instruksjon [8].

En utfordring ved bruk av LLM'er er at modellene krever betydelige datamengder, regnekraft og energi for både trening og bruk. I tillegg kommer etiske hensyn, inkludert risikoen for å reprodusere skjevheter i treningsdataene, samt spørsmål rundt transparens og kontroll over modellens prediksjoner. Dette har ført til økt interesse for teknikker som kontrollert generering, fine-tuning og alignment for å sikre at modellen handler i tråd med ønskede verdier og bruksmål.

2.5 Transkribering av tale til tekst og talegjenkjenning

Transkribering er prosessen der muntlig språk konverteres til skriftlig representasjon. I informatikk og språkteknologi refererer dette vanligvis til automatisk talegjenkjenning (Automatic Speech Recognition, ASR), hvor digitale systemer analyserer lydsignaler for å identifisere og tolke språklig innhold [1]. Dette utgjør et viktig grensesnitt mellom menneskelig kommunikasjon og datamaskinbasert prosessering, og danner grunnlaget for videre behandling i mange språkteknologiske applikasjoner.

Formålet med transkribering strekker seg utover ren dokumentasjon; det handler om å gjøre lydinnhold tilgjengelig for søk, strukturert analyse og semantisk tolkning. Transkriberte data gjør det mulig å bruke tekstbaserte metoder innenfor informasjonshenting, tekstanalyse og maskinlæring. Dette er særlig viktig i systemer som kombinerer ASR med naturlig språkprosessering (NLP), for eksempel i virtuelle assistenter, automatiserte kundeservicesystemer og i samtaleanalyse [9].

Den automatiserte transkripsjonsprosessen består typisk av flere komponenter: først en signalprosessering som konverterer det akustiske signalet til en sekvens av funksjoner; deretter en akustisk modell som estimerer sannsynlighetene for språklige enheter (som fonemer); og til slutt en språkmodell og dekoder som sammen genererer den mest sannsynlige tekstlige representasjonen av den opprinnelige talen. Moderne systemer er gjerne bygget på dype nevrale nettverk, som for eksempel rekurrente nevrale nettverk (RNN), Long Short-Term Memory-nettverk (LSTM) eller transformerarkitekturer, og bruker ofte teknikker som attention-baserte sequence-to-sequence-modeller [10, 7].

I vår kontekst fungerer tale-til-tekst-transkribering som en kritisk forprosess. Den etablerer en tekstlig inngang til systemet som videre benyttes i informasjonshenting, semantisk tolkning og generering av svar fra språkmodeller. Transkripsjonsnøyaktighet er derfor essensiell: feil i tidlige ledd kan propagere gjennom systemet og redusere både ytelse og brukertilfredshet. Dette gjelder spesielt i applikasjoner der output fra en språkmodell er direkte avhengig av

kvaliteten på transkribert input.

2.6 Finjustering av store språkmodeller

Finjustering (fine-tuning) refererer til prosessen med å tilpasse en forhåndstrent språkmodell til en spesifikk oppgave eller et avgrenset domene. Store språkmodeller (LLMs) er i utgangspunktet trent på enorme mengder generalisert tekstdata og besitter bred kunnskap og språklig kompetanse. For å øke ytelsen på oppgavespesifikke problemer – som juridiske tekster, medisinske diagnoser eller samtaler i kundeservice – kan modellen videre trenes på et mindre datasett som representerer den nye bruken. Dette gjør det mulig for modellen å tilpasse seg både innholdsmessig og språklig kontekst, uten å måtte trenes helt fra bunnen av.

Finjustering skjer typisk ved at modellens parametere oppdateres gjennom videre gradientbasert optimalisering på det nye datasettet. Sammenlignet med full pretrening er dette langt mer ressursbesparende, siden det kun krever en brøkdel av data og regnekraft [11]. I motsetning til prompt engineering og in-context learning, hvor modellen holdes uendret og konteksten manipuleres, endres de interne vektverdiene ved finjustering – noe som kan gi vedvarende forbedringer i ytelsen på spesifikke domener eller oppgaver.

Ulike former for finjustering eksisterer. Full finjustering innebærer at hele modellen optimaliseres på nytt, mens mer moderne tilnærminger som parametereffektiv finjustering (PEFT) benytter teknikker som LoRA (Low-Rank Adaptation) eller adapterlag for å oppdatere bare deler av modellen [12]. Dette gjør det mulig å tilpasse svært store modeller, selv med begrenset maskinell kapasitet.

I mange anvendelser – særlig der personvern, datasikkerhet eller domeneavhengighet er viktig – representerer finjustering et effektivt kompromiss mellom generaliserbarhet og spesialisering. Det muliggjør også kontinuerlig læring, der modellen gradvis oppdateres etter hvert som ny data blir tilgjengelig.

2.7 Information Retrieval (IR)

Information Retrieval (IR) handler om prosessen med å finne relevant informasjon i et større datasett, basert på en spesifikk informasjonsforespørsel. Dette er et kjerneområde innen informatikk, og har tradisjonelt blitt brukt i systemer som søkemotorer (f.eks. Google, Bing og DuckDuckGo) for å hente ut dokumenter eller nettsider som er relevante for brukernes søk [13].

I konteksten av store språkmodeller (LLM-er) har IR fått en ny betydning. Her brukes IR for å hente relevant kontekst fra eksterne kunnskapskilder, slik at modellen kan gi mer presise og faktabaserte svar. Denne integrasjonen mellom IR og LLM kalles ofte "retrieval-augmented generation", og er grunnlaget for teknikker som RAG (Retrieval-Augmented Generation), CAG (Contrastive Augmented Generation) og ReACT (Reasoning and Acting). Disse teknikkene kombinerer informasjonsinnhenting med generative modeller for å produsere svar som er både språklig sammenhengende og kunnskapsrikt fundert [2].

2.7.1 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) er en metode innenfor naturlig språkprosessering som kombinerer henting av relevant informasjon fra eksterne kunnskapskilder med generative språkmodeller. På norsk kan man kalle metodikken for *hente-forsterket generering.*

Formålet med RAG er å øke presisjon og faktabasertheten i genererte svar ved å forsyne språkmodellen med kontekstbasert informasjon fra pålitelige kilder. Dette vil være med å redusere risikoen for hallusinasjoner i genererte svar - tilfellet der språkmodellen genererer feilaktig eller oppdiktet informasjon når den mangler nødvendig kunnskap om et emne, eller en sak i vårt tilfelle [2].

RAG-arkitekturen består typisk av to hovedkomponenter: en henter (retriever) og en generator. Henteren bruker en encoder-transformer til å konvertere en brukers forespørsel (query) til en vektorrepresentasjon. Denne vektoren sammenlignes så med representasjoner i en ekstern kunnskapsdatabase, ofte en vektordatabse, for å finne semantisk relevante dokumenter. Vektorbaserte søk benytter teknikker som kosinus-similaritet eller euklidisk avstand for å sammenligne semantiske likheter mellom vektorer [14].

Fordelen med vektorbaserte søk fremfor tradisjonelle nøkkelordsøk er selve evnen til å utføre *semantisk matching.* Dette betyr i hovedsak at søket ikke er begrenset til eksakte nøkkelordtreff, men også identifisere konsepter og meninger som er språklig ulike, semantisk like. For eksempel i vårt tilfelle, vil dette gi muligheten å koble en feilmelding som beskriver: "*nettverksproblemer på legevakten*" med tidligere feilmeldinger som "*ingen wifi i helsetjenesten*", selv om ordvalget er helt forskjellig - basert på semantisk likhet.

Når relevante dokumenter er hentet, sender RAG-systemer disse sammen med den opprinnelige forespørselen til den generative komponenten. Språkmodellen har da mulighet å generere et svar som tar hensyn til brukerens spørsmål, og den innhentede kontekstuelle informasjonen. Dette vil resultere i et svar som ikke bare er språklig velformulert, men også forankret i faktisk kunnskap fra de eksterne kildene.

For Bodø kommune sitt tilfelle kan RAG brukes til å klassifisere innkommende feilmeldinger etter kritikalitet, identifisere lignende historiske hendelser, og utarbeide løsingsforslag og klassifisering basert på tidligere erfaringer. Dette gjør RAG systemet spesielt verdifullt for IKT-avdelinger som håndterer gjentakende problemer på tvers av ulike systemer og tjenester

2.7.2 Cache Augmented Generation (CAG)

Cache Augmented Generation (CAG) er en metode som ligner på Retrieval-Augmented Generation (RAG) ved at den benytter eksterne datapunkter for å forbedre en spørring til en stor språkmodell (LLM). Forskjellen mellom de to tilnærmingene ligger primært i hvordan de integrerer disse dataene. I CAG blir relevante datapunkter forhåndsinnlastet i LLM-en og integrert direkte i kontekstvinduet, hvilket betyr at all informasjon som trengs for å generere svaret, er tilgjengelig i modellen på forhånd. På den andre siden benytter RAG en retrieval-prosess, hvor relevant innhold hentes fra en ekstern kunnskapsdatabase (ofte en vektordatabase) ved hjelp av semantisk eller nøkkelord-basert søk.

En betydelig fordel med CAG er at det eliminerer behovet for retrieval-fasen, noe som gjør prosessen raskere, ettersom dataene allerede er lastet inn i modellen. Imidlertid er dette også en begrensning, ettersom metoden er begrenset til den tilgjengelige informasjonen som kan passe inn i kontekstvinduet til modellen. I motsetning til CAG, kan RAG hente inn eksterne data uten å være begrenset av modellens kontekstvindu, og dermed håndtere større datamengder. Likevel kan RAG være tregere enn CAG, ettersom den krever tid til å hente relevant informasjon fra databasen [15].

2.7.3 ReAct (Synergizing Reasoning and Acting in Language Models)

ReAct (Reasoning and Acting) er en tilnærming som kombinerer eksplisitt resonnering med målrettede handlinger for å forbedre språkmodellers evne til å løse komplekse oppgaver som krever både kunnskapsinnhenting og beslutningstaking. I motsetning til mer tradisjonelle Retrieval-Augmented Generation (RAG)-metoder, som utelukkende fokuserer på å hente informasjon før generering, legger ReAct vekt på en iterativ prosess der modellen resonnerer over tidligere kontekst og aktivt velger handlinger som påvirker videre informasjonsflyt.

I denne modellen genererer LLM-en ikke bare et svar direkte, men produserer en blanding av tanker (thoughts) og handlinger (actions) i et vekselsvis mønster. Tankene reflekterer eksplisitt resonnement, mens handlingene involverer forespørsler til eksterne verktøy, for eksempel kunnskapsbaser eller søkemotorer. Denne mekanismen gjør det mulig for modellen å lære mens den utfører oppgaven, noe som gir mer informerte og korrekte beslutninger over tid [16]. Selv om ReAct-metoden har vist seg å være spesielt effektiv i oppgaver som spørsmålsbesvarelse og problemløsning, er det kanskje tenkelig at metodens egenskap å lære underveis, kan føre til komplikasjoner dersom vi har mye forhåndsdefinerte betingelser, eller spesifikke krav for en feilmelding - da kanskje tilnærmingen, uten fintuning kan lære unødvendig informasjon som kanskje ikke er nødvendig å lære.

ReAct-metoden har vist seg å være spesielt effektiv i oppgaver som krever multi-hop reasoning, spørsmålsbesvarelse og problemløsing, der både presis informasjonsinnhenting og logisk tenkning er avgjørende. I slike scenarier har ReAct-modeller oppnådd bedre ytelse enn modeller som kun bruker enten ren generering eller tradisjonell informasjonshenting alene.

2.8 Algoritmer

2.8.1 Auto-regressiv dekoding

Auto-regressiv dekoding er en grunnleggende teknikk i generative språkmodeller der hvert påfølgende token predikeres sekvensielt basert på tidligere genererte tokens [17]. Den underliggende sannsynlighetsmodellen er gitt ved:

$$P(w_{1:T} \mid W_0) = \prod_{t=1}^{T} P(w_t \mid w_{1:t-1}, W_0) \tag{1}$$

hvor:

- $W_0 \in \mathbb{R}^d$ er startkonteksten (input embedding)
- $w_{1:t-1} = (w_1, \ldots, w_{t-1})$ er den akkumulerte tokensekvensen
- $w_t \in \mathcal{V}$ er neste token i vokabularet $\mathcal{V}$
- T er sekvenslengden (variabel)

Med initialbetingelsen $w_{1:0} = \varnothing$. Genereringen avsluttes når:

- EOS-token (end-of-sequence) genereres, dvs. $w_t =$ ¡EOS¿
- T når maksimal lengde $T_{\max}$

Tokenseleksjonsmetoder Ved hvert tidstrinn t velges w_t ved en av disse metodene:

1. **Grådig søk (Greedy search)**:
   Greedy search (grådig søk) er en dekoderingsmetode som ved hvert tidspunkt t velger det mest sannsynlige token w_t basert på den betingede sannsynlighetsfordelingen [17]. Formelt kan dette uttrykkes som:

$$w_t = \underset{w \in \mathcal{V}}{\operatorname{argmax}} P(w \mid w_{1:t-1}, W_0) \tag{2}$$

   Metoden har følgende egenskaper:

   - **Deterministisk**: Alltid gir samme utputt for gitt input
   - **Effektiv**: Krever minimal beregningskraft per steg
   - **Begrenset**: Kan føre til suboptimale sekvenser på grunn av manglende tilbakesporing

   Eksempel: Gitt en sannsynlighetsfordeling $P(w|\text{"Norge er et"}) = \{\text{"land"} : 0.6, \text{"vann"} : 0.3, \text{"fjell"} : 0.1\}$, vil greedy search alltid velge tokenet *"land"*.

   Et problem med grådig søk (greedy search) er at det kun velger det mest sannsynlige token på hvert steg, og dermed kan overse globale optimale sekvenser der det først må velges et token med lavere sannsynlighet for å nå bedre utfall senere. Dette gjør at metoden ofte overser "skjulte" sekvenser med høy samlet sannsynlighet [17].

   Greedy search brukes ofte i talegjenkjenning (ASR)-modeller som en rask og enkel dekoderingsmetode. Den gir lav beregningskostnad og lav latens ved å velge det mest sannsynlige token i hvert steg uten å ta hensyn til alternative sekvenser. Imidlertid finnes det i dag mer nøyaktige, men også mer ressurskrevende metoder som beam search.

2. **Strålesøk (Beam search)**:
   I motsetning til *greedy search*, som kun velger det mest sannsynlige tokenet ved hvert steg, opprettholder beam search flere alternative sekvensruter parallelt. Ved å holde de k mest sannsynlige delsekvensene i hvert steg, kan beam search utforske et bredere søkerom og dermed identifisere sekvenser med høyere samlet sannsynlighet. Algoritmen kan betraktes

som en bredde-først søkestrategi, der man på hvert nivå i søketreet beholder de k beste kandidatene basert på deres akkumulerte sannsynlighet [17].

Beam search holder de k mest sannsynlige sekvenshypotesene ved hvert tidstrinn t, der k er strålestørrelsen (ofte kalt *beam width*). Formelt uttrykkes dette som:

$$\mathcal{B}_t = \underset{w_{1:t}}{\text{top-}k} \prod_{i=1}^{t} P(w_i \mid w_{1:i-1}, W_0) \tag{3}$$

Når $k = 1$, reduseres beam search til greedy search.

Beam search brukes mye i automatisert talegjenkjenning (ASR) for å finne den mest sannsynlige transkripsjonen av gitt lyddata. Ved å opprettholde flere mulige sekvenshypoteser samtidig, gjør beam search det mulig å korrigere for usikkerhet og flertydighet i de akustiske signalene, og gir derfor bedre resultater enn deterministiske metoder som grådig søk [18].

3. **Stokastisk utvalg (Sampling)**:

   - **Vanlig sampling**: Neste token velges tilfeldig fra hele sannsynlighetsfordelingen:

     $$w_t \sim P(w \mid w_{1:t-1}, W_0)$$

     Dette gir variasjon i generert tekst, men kan føre til valg av mindre sannsynlige eller irrelevante ord.

   - **Top-k sampling**: Man begrenser utvalget til de k mest sannsynlige tokenene, og normaliserer sannsynlighetene:

     $$w_t \sim P_k(w \mid w_{1:t-1}, W_0), \quad \text{der } P_k(w) = \begin{cases} \frac{P(w)}{\sum_{w' \in V_k} P(w')} & \text{hvis } w \in V_k \\ 0 & \text{ellers} \end{cases}$$

     hvor V_k er mengden av de k mest sannsynlige tokenene.

   - **Top-p sampling (nukleus sampling)**: Her velges et dynamisk antall token slik at summen av deres sannsynligheter overstiger terskelen p:

     $$V_p = \{w_1, w_2, \ldots, w_n\} \quad \text{slik at } \sum_{i=1}^{n} P(w_i) \geq p$$

     Deretter samles det fra den normaliserte fordelingen over V_p:

     $$w_t \sim P_p(w \mid w_{1:t-1}, W_0)$$

   - **Temperaturjustering**: Før sampling kan sannsynlighetene justeres med en temperaturparameter $T > 0$:

     $$P_T(w) = \frac{P(w)^{1/T}}{\sum_{w'} P(w')^{1/T}}$$

     Lav temperatur ($T < 1$) gjør fordelingen skarpere (mer deterministisk), mens høy temperatur ($T > 1$) gjør den flatere (mer variert).

2.8.2 Semantisk søk

Semantisk søk refererer til metoder for å hente informasjon basert på betydningen bak ordene, i motsetning til tradisjonelle søkemetoder som kun er basert på tekstuelle mønstre. Flere algoritmer benyttes for å beregne semantisk likhet mellom spørsmålet og dokumentene i et datasett. Her beskrives noen av de mest relevante algoritmene.

K-nearest-neighbour (KNN) Algoritme: K-nearest-neighbour (KNN) algoritmen er en populær metode for å finne de nærmeste naboene til et punkt i et datasett. Algoritmen fungerer ved å tildele en verdi til et punkt basert på de fleste etikettene blant de nærmeste punktene i datasettet [19]. KNN er enkel å implementere og krever minimal trening, men kan være beregningsmessig tungvint for store datasett, da den krever at avstanden mellom hvert punkt må beregnes for hver forespørsel.

En stor fordel med KNN er at den kan brukes til både klassifisering og regresjon, avhengig av typen problem. I sammenheng med Retrieval-Augmented Generation (RAG) kan KNN brukes til å hente relevante informasjonspunkter fra en vektor-database, som kan benyttes til å forbedre modellen ved å gi kontekst eller fakta [20].

Cosinuslikhet (Cosine Similarity): Cosinuslikhet er en av de mest brukte metodene for å beregne semantisk avstand mellom dokumenter eller ordvektorer. Algoritmen benytter seg av vinkelen mellom to vektorer for å vurdere deres likhet [21, s. 299–301]. Dette er særlig nyttig når man arbeider med høydimensjonale rom som de som genereres av ordvektormodeller. Cosinuslikhet er effektiv i situasjoner hvor man sammenligner tekstbaserte representasjoner av et stort datamengde, spesielt når de er representert som vektorer.

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$$

Der A og B er vektorer for de to dokumentene.

I RAG kan cosinuslikhet brukes til å sammenligne spørsmålet (eller input) med dokumenter i en vektor-database, og hente de mest relevante dokumentene basert på deres semantiske likhet til spørsmålet.

Dot Produkt: Dot-produktet er en grunnleggende matematisk operasjon som brukes til å beregne likheten mellom to vektorer. Når to vektorer representerer dokumenter eller ord i et semantisk rom (for eksempel generert ved hjelp av ordembeddings), kan dot-produktet brukes til å vurdere hvor like disse representasjonene er.

Formelen for dot-produktet mellom to vektorer A og B er:

$$A \cdot B = \sum_{i=1}^{n} A_i B_i$$

Der A_i og B_i er komponentene til vektorene A og B, og n er antall komponenter i vektorene. Resultatet av dot-produktet er et enkelt skalarprodukt som reflekterer graden av likhet mellom de to vektorene. Hvis dot-produktet er høyt, betyr det at de to vektorene er mer like i forhold til sine semantiske innhold [22].

I sammenheng med semantisk søk og Retrieval-Augmented Generation (RAG) kan dot-produktet brukes til å finne de mest relevante dokumentene for et gitt spørsmål ved å beregne likheten mellom spørsmåls-vektoren og dokument-vektorene i en database. En høy dot-produktverdi indikerer at spørsmålet og dokumentet har en sterk semantisk likhet, og dermed er dokumentet relevant for spørsmålet.

Et alternativt mål som ofte brukes i kombinasjon med dot-produktet, er **cosinuslikhet**, som normaliserer vektorene før beregning for å unngå at lengden på vektorene påvirker resultatet.

Dot-produktet kan være en av mange mulige algortimer for å hjelpe med å hente informasjon i RAG-modeller.

K-means Clustering: K-means er en metode for dataklynging som deler et datasett i K grupper basert på likheten mellom elementene. I semantisk søk benyttes K-means for å gruppere dokumenter basert på deres semantiske innhold. Etter at dokumentene er delt inn i klynger, kan man hente de mest relevante klyngene for et gitt spørsmål og deretter analysere dokumentene innenfor disse klyngene [23].

I RAG kan K-means brukes til å gruppere dokumenter i semantiske klynger og hente relevant informasjon fra den mest passende klyngen.

2.9 Skytjenester og virtualisering

I moderne programvaresystemer har bruk av skytjenester blitt en sentral strategi for å håndtere krav om skalerbarhet, tilgjengelighet og kostnadseffektivitet. Skytjenester innebærer levering av ressurser og tjenester – som lagring, nettverk, databehandling og programvare – over internett. Dette kan klassifiseres i tre hovedkategorier: infrastruktur som tjeneste (IaaS), plattform som tjeneste (PaaS) og programvare som tjeneste (SaaS) [24]. Slike tjenester gjør det mulig å tilpasse ressursbruk dynamisk, noe som er spesielt gunstig for systemer med varierende belastning eller vekstpotensial.

En sentral egenskap ved skytjenester er elastisitet, det vil si evnen til å automatisk tilpasse seg endringer i ressursbehov, samt støtte for geografisk distribusjon og feiltoleranse. Dette gjør skymiljøer godt egnet for utvikling og drift av både mikrotjenestebaserte og monolittiske systemer.

2.10 Containere og applikasjonsisolasjon

Containere er en virtualiseringsteknologi som tilbyr et lettvekts og konsistent kjøremiljø for applikasjoner. I motsetning til tradisjonell maskinvirtualisering, deler containere operativsystemkjernen med vertssystemet, men isolerer brukermiljøet. Dette gir rask oppstartstid og lavt ressursforbruk, noe som er

særlig nyttig i utviklings- og produksjonsmiljøer der effektiv ressursutnyttelse er avgjørende [25].

Containere fremmer reproduserbarhet ved å pakke applikasjonskode, avhengigheter og konfigurasjoner i én enhet. Dette reduserer risikoen for avvik mellom utviklings-, test- og produksjonsmiljøer. I kombinasjon med orkestreringssystemer muliggjør containere automatisert distribusjon, lastbalansering, feilhåndtering og skalerbarhet i store distribuerte systemer [26].

I tillegg forenkler containere distribusjon til skyplattformer ved å tilby et isolert og forutsigbart kjøremiljø, noe som gjør dem velegnet for moderne applikasjonsutvikling.

2.11 Distribuerte systemer og automatisering

Bruken av skybaserte miljøer og containere henger tett sammen med prinsipper fra distribuerte systemer, hvor ulike komponenter opererer på flere maskiner, men samarbeider som én enhet. Effektiv styring av slike systemer krever støtte for tilstandshåndtering, nettverkskommunikasjon og ressurskoordinering. I tillegg gir skytjenestemodeller støtte for infrastruktur som kode og kontinuerlig integrasjon og distribusjon (CI/CD), som muliggjør automatisert og repeterbar utrulling av programvare.

Disse teknologiene muliggjør utvikling av robuste og skalerbare løsninger som kan tilpasses både små og store brukerbaser. I tillegg forenkles eksperimentering og testing, særlig i sammenhenger der maskinlæring eller datadrevet funksjonalitet krever hyppige iterasjoner og distribusjon av nye modeller.

2.12 Utviklingsmetodikk

2.12.1 Scrum

Scrum er en utviklingsmetode som ble utformet av Ken Schwaber og Jeff Sutherland på starten av 1990-tallet [27]. Metoden deler opp utviklingsperioden i mindre tidsintervall, så kalte sprinter, hvor hver sprint har tydelig struktur. Teamet i et Scrum-miljø bør være stort nok til å kunne håndtere rollene og oppgavene som kreves av metoden, men samtidig liten nok til at oppgavene blir løst med smidighet og fleksibilitet, gjerne 5-11 personer [28].

Metoden bruker definerte roller i utviklingsteamet, hvor en Scrum Master er en slags prosjektleder og har ansvaret for fremgang i utviklingen og at scrummetodikken blir fulgt; Product Owner har ansvaret for at produktet blir utviklet, og bestemmer hvilke deler av produktet som skal utvikles til hver sprint; til slutt er utviklerne hoveddelen av teamet.

2.12.2 Scrumban

Scrumban er en hybrid utviklingsmetodikk som kombinerer elementer fra både Scrum og Kanban. Metoden ble utviklet for å gi team større fleksibilitet enn det Scrum tradisjonelt tilbyr, samtidig som man beholder noen av Scrum sine strukturer som sprintplanlegging og regelmessige møter. I Scrumban brukes Kanbans visuelle oppgavekort og flytprinsipper for å forbedre arbeidsflyten, mens Scrum

sine roller og iterasjoner kan beholdes, men ofte med mer dynamiske sprintlengder. Dette gjør det enklere å tilpasse seg endringer i krav og prioriteringer uten å miste fokus på kontinuerlig forbedring og levering [29].

Hovedmomentet med Scrumban er at det bruker deler fra både Scrum og Kanban, men lar teamet bestemme hvilke deler som skal brukes og til hvilken grad. Dette kan være en fordel siden det bidrar til å gjøre utviklingen mer smidig og lar teamet få skreddersy utviklingen etter sine behov. Ulempen er at denne friheten kan påvirke produktiviteten dersom strukturen blir for svak. Scrumban forutsetter at utviklerne har kjennskap med både Scrum og Kanban siden metoden deler elementer fra begge metodene.

2.12.3 Kanban

Kanban er en visuell metode for å styre arbeidsprosesser og oppgaver, opprinnelig utviklet innen produksjonsindustrien, men tilpasset programvareutvikling og annet kunnskapsarbeid. I Kanban visualiseres arbeidsflyten ved hjelp av et tavlesystem med kolonner som representerer ulike faser i prosessen (for eksempel "To Do", "In Progress" og "Done"). Arbeidsoppgaver flyttes gjennom kolonnene etter hvert som de utføres. En sentral idé i Kanban er å begrense antallet oppgaver som pågår samtidig (WIP – Work In Progress), for å sikre flyt og redusere flaskehalser. Kanban har ingen faste roller eller tidsbegrensede sprinter, og fokuserer derfor på kontinuerlig levering og forbedring [30].

Et viktig moment i Kanban er at hver kategori har en Work-in-Progress (WIP) begrensning. Kort forklart bestemmer WIP hvor mange oppgaver som kan være i hver kategori. Begrensningen bidrar til at de oppgavene som er påbegynt blir ferdige, og forhindrer at utviklerne starter med for mange forskjellige oppgaver om gangen. Det er viktig at utviklerne følger med på WIP-begrensningen og gjør oppgavene i de forskjellige kategoriene, for å forhindre at det blir en blokade i en av kategoriene som forhindrer framgang i utviklingen.

Kanban deler ikke inn i utviklingsperioder, men lar heller WIP begrensningen styre hvilke oppgaver som arbeides med. Begrensningen oppfordrer utviklerne til å dele arbeidet opp i mindre stykker som kan bevege seg raskt gjennom tavlen og til ferdig-kategorien. Det er heller ingen prioriteringer av oppgaver slik som i Scrum, og det betyr at utviklerne selv kan bestemme og endre prioritet fortløpende.

3 Valg av teknologi og utviklingsmetode

3.1 Valg av utviklingsmetodikk

Vi valgte å bruke Kanban som utviklingsmetodikk fordi metodens fokus på smidig planlegging og lave krav til dokumentasjon gjorde at vi kunne bruke mer tid til utviklingen av produktet. Størrelsen på gruppen var en faktor til at valget falt på Kanban, siden Kanban ikke krever at man har planleggingsøkter eller dedikerte roller. Dersom vi hadde brukt Scrum eller Scrumban hadde det betydd mye mer planlegging fram mot hver sprint, og oppfølging etter endt sprint. I en større gruppe hvor man kunne hatt en dedikert Scrum Master og Product Owner, er Scrum en metode som kan gi bra resultater, men i en liten gruppe som vår ble det for store krav til hvert enkelt medlem og for rigid struktur. I tillegg er kravene til dokumentasjon veldig høye i Scrum, med blant annet Sprint planlegging og retrospective.

Kanbans bruk av en tavle til å planlegge utviklingen, gir en enkel måte å følge med på hva som gjøres av hvert gruppemedlem, og bidrar til å holde oversikt over utviklingsarbeidet. Selv om alle tre medlemmene i gruppen bor i Bodø, bor vi et stykke fra hverandre og vi jobbet for det meste alene eller digitalt. Derfor var det viktig å kunne sjekke hva som ble jobbet med til enhver tid. Vi vektla også metodens smidighet høyt, fordi det ga oss mulighet til å endre rekkefølge på arbeidsoppgaver. I et sammensatt prosjekt som det vi har arbeidet med, har det vært flere hendelser hvor vi har vært nødt til å endre på planlagt rekkefølge av utviklingen, fordi vi har oppdaget avhengigheter eller mangler i løpet av arbeidet.

3.2 Valg av hovedprogrammeringsspråk

I utviklingen av løsningen har vi hovedsakelig benyttet Python som hovedprogrammeringsspråk. Dette valget er basert på vår tidligere erfaring med språket, samt at Python gir enkel og effektiv formulering av koden, noe som forenkler både utvikling og vedlikehold av systemet. I tillegg er Python godt egnet for oppgaver som databehandling, enkle syntaks, og har god integrasjon med Azure-tjenester, samt god og enkel tilgjengelig dokumentasjon - som er sentralt i vår løsning. Med tanke på vedlokeholdbarhet og videreutvikling av systemet, og for tekniske detaljer har vi utarbeidet en systemdokumentasjon (4) som beskriver teknologistack, kode og arbeidsflyt.

Vi har også brukt JavaScript i mindre grad, spesielt i utviklingen av en webapplikasjon for klassifisering av feilmeldinger, samt simulering av telefonsamtaler for å minke kostnaden til telefon samtaler.

3.3 Bruk av pre-trente modeller

I dette prosjektet benyttes pre-trente modeller som NB-Whisper for norsk transkribering og en RAG-løsning basert på nevralnettverk. Disse modellene er forhåndstrent på store datasett ved hjelp av dype nevralnettverk, noe som muliggjør nøyaktige transkripsjoner og generative svar uten behov for å trene modellene fra bunnen av. Pre-trente modeller har til fordel gjennom overføringslæring, som vil si modeller som allerede er trent på store datasett, har mulighet for å

finjusteres til spesifikke oppgaver. En annen fordel er datamengder og regneressurser er betydelig mindre enn det som ville vært nødvendig, dersom man skulle trent en modell fra bunnen av.

3.4 Klassifisering – teknologivalg

For klassifiseringen er det foreslått fra Bodø kommune å benytte Retrieval Augmented Generation (RAG) som metode. Hovedfordelen med RAG er at vi kan gi den store språkmodellen (LLM) kontekst og informasjon fra eksterne kilder, altså utenfor modellens forhåndstrente kunnskapsbase. Dette gjør det mulig å tilpasse svarene til det aktuelle domene og hente ut anbefalinger spesifikt relatert til innmeldte feilmeldinger.

For å støtte klassifiseringen har vi integrert dokumenter som kritikalitetsmatrisen og lagring av historiske feilmeldinger. Disse kildene benyttes som inndata til RAG-systemet for å gi modellen nødvendig kontekst, og sikre en mer presis vurdering.

3.4.1 RAG skytjeneste løsning

RAG-løsningen er implementert som en skytjeneste i Azure, der vi benytter *Azure AI Search* i kombinasjon med en vektordatabase. Kritikalitetsmatrisen og den historiske databasen vektoriseres og lagres i denne databasen, slik at relevant informasjon raskt kan hentes ut og benyttes som kontekst for språkmodellen. Ved å bruke vektorbasert søk oppnår vi semantisk forståelse av feilmeldingene, noe som gjør systemet i stand til å identifisere og foreslå relevante tidligere hendelser og klassifiseringer på en mer treffsikker måte.

3.4.2 Semantisk søk og klassifisering – algoritme

Algoritmen bak vår løsning for semantisk søk og klassifisering av feilmeldinger bygger på en hybrid tilnærming som kombinerer nøkkelordbasert søk og semantisk vektorsøk.

Den nøkkelordbaserte komponenten benytter klassiske teknikker som TF-IDF eller BM25 for å hente dokumenter som inneholder eksakte eller delvise treff basert på brukerens forespørsel. Parallelt gjennomføres et semantisk vektorsøk, der både brukerens feilmelding og dokumentene representeres som vektorer i et flerdimensjonalt rom. Til dette benyttes OpenAI-modellen `text-embedding-3-large`, som genererer høykvalitets tekstrepresentasjoner.

Vårt hybridsøk implementeres direkte i Azure AI Search, hvor vi kombinerer tekstøking med vektorbasert søk i samme spørring, noe som gir skal gi oss mer presise resultater, kontra å anvende kun én av de. Denne implementeringen tillater systemet vårt å finne relevante dokumenter, selv når bruken av begreper varierer betydelig mellom feilmeldingene.

Vektorsøket benytter nærmeste nabo-søk (k-NN) for å identifisere dokumenter med høy semantisk likhet til brukerens input. Dette muliggjør treff på relevante dokumenter selv når nøkkelordene ikke samsvarer eksplisitt.

Resultatene fra begge metodene kombineres deretter i en hybrid rangeringsalgoritme, som vekter både semantisk nærhet og nøkkelordrelevans. Denne funksjo-

nen er innebygd i Azures AI Search tjeneste, og vil kunne refereres i form av en "relevans" variabel, også kalt "@serch.score" som returneres med hvert resultat. For vektorbaserte søk ligger denne verdien normalt mellom 0-1, hvor høyere tall indikerer større likhet.

Fordelen dette gir oss, er en god løsning for klassifisering og anbefaling av feilmeldinger, basert på tidligere kjente problemstillinger – også når formuleringene varierer betydelig.

3.5 Feilforslag

For å generere relevante forslag til mulige årsaker og løsninger på innrapporterte feil, benyttes en RAG-baserte tilnærming (Retrieval-Augmented Generation). Ved hjelp av denne metoden kombineres søk i en dokumentbase med generativ kunstig intelligens.

Når en bruker sender inn en feilmelding via skjemaet eller telefon, utløses en prosess der nøkkelinformasjon fra meldingen brukes til å søke i en forhåndsindeksert dokumentkilde. Dokumentene kan inneholde tidligere feilmeldinger, teknisk dokumentasjon eller FAQ-er. De mest relevante dokumentene hentes og gis som kontekst til en språkmodell som genererer forslag til mulige feilkilder eller løsninger, som deretter presenteres for brukeren eller administrator.

Denne tilnærmingen gjør det mulig å tilby mer presise og kontekstuelt relevante forslag enn tradisjonelle søkebaserte eller regelbaserte systemer. RAG gir dermed et mer intelligent beslutningsstøttesystem, spesielt nyttig i feilhåndteringsprosesser hvor det finnes store mengder historiske data.

3.6 Skytjenesteløsning

Vurderingen av skytjenester, som beskrevet i forprosjektet (2) vårt, ledet oss til valg av Azure. Selv om det finnes flere leverandører av skytjenester, som Amazon Web Services og Google Cloud, valgte vi Azure på grunn av det brede utvalget av integrerte tjenester som dekker våre behov, samt fordi Bodø kommune allerede benytter Azure i sin IKT-infrastruktur. Dette sikret god kompatibilitet og enklere integrasjon med eksisterende systemer.

Valget av Azure ble også påvirket av gruppens tidligere erfaring med plattformen. I prosjektet har vi benyttet flere Azure-tjenester, inkludert Function App, Container Instances, Container Apps, Azure AI Search, Azure AI Foundry, Communication Services og Storage Account, for å bygge en fleksibel og skalerbar løsning tilpasset problemstillingen.

3.6.1 Function Apps

Den nåværende skyløsningen består av fire separate Function Apps, der hver er dedikert til en spesifikk oppgave. Denne inndelingen ble valgt under utviklingen for å forenkle testing og opprettholde modularitet. De ulike Function Appene er vist i Tabell 1.

Hver av disse Function Appene benytter en egen tilknyttet Storage Account for lagring av WebJobs-data. Dette valget ble gjort for å skille logger og annen

Function App	Oppgave
`feilmelding-prosessering`	Prosessering av transkriberte feilmeldinger til bruk i RAG
`opptak-t-bachelor2025`	Telefoni og lydopptak
`skjema-t-bachelor2025`	Prosessering av skjema
`transkribering-t-bachelor2025`	Transkribering av lydopptak

Table 1: Oversikt over Function Apps og deres oppgaver

funksjonsspesifikk informasjon fra hverandre, noe som gjør utvikling, feilsøking og vedlikehold enklere.

3.6.2 Storage Accounts

Løsningen benytter for øyeblikket fire forskjellige Storage Accounts. To av disse er automatisk generert av Azure Function-appene:

- `rgtbachelor20259b8c` – generert for `transkribering-t-bachelor2025`
- `rgtbachelor20258cac` – generert for `skjema-t-bachelor2025`

I tillegg benyttes følgende manuelt opprettede Storage Accounts:

- `stfeilmelding001` – brukes av `feilmelding-prosessert`
- `stopptak001` – brukes av `opptak-t-bachelor2025`

3.6.3 Azure Container Registry

Løsningen benytter også Azure Container Registry (ACR) for å lagre og distribuere containerbaserte, isolerte applikasjoner. Dette gjør det mulig å trekke ned containere, som for eksempel transkriberingsmoduler, uten begrensninger knyttet til antall nedlastinger. Ved å bruke ACR sikres høy tilgjengelighet og kontroll over versjonering og tilgangsstyring av containerbaserte applikasjoner i løsningen.

3.6.4 Azure Container Apps eller Container Instances

Det er også lagt til støtte for å bruke enten Azure Container Apps (ACA) eller Azure Container Instances (ACI), avhengig av hvilken kommando som gis til containeren ved oppstart.

Dersom transkriberings containeren startes med kommandoen `--run-webapp`, konfigureres den som en webapplikasjon, og kjøres typisk i ACA for å dra nytte av autoskalering og HTTP-endepunkter.

Hvis containeren i stedet startes med `--use-call-recording`, er dette scenariet bedre egnet for ACI, som kan brukes for kortvarige, batch-orienterte oppgaver knyttet til behandling av lydopptak.

Begge disse metodene er utviklet for å behandle JSON-basert data som inneholder all relevant informasjon relatert til en telefonhenvendelse, inkludert metadata og lydopptak. Dette gjør det mulig å standardisere behandling på tvers av ulike driftsmiljøer og bruksområder.

3.6.5 Arkitektur for transkriberingstjenesten

Ved bruk av Azure Container Apps (ACA) håndteres transkriberingsprosessen ved at en HTTP POST-forespørsel sendes til containerapplikasjonen. Containeren returnerer deretter transkriberingen sammen med tilhørende metadata som en del av responsen. I tillegg lagres denne informasjonen i en Azure Blob Storage-container. Dette lagringssteget utløser en hendelse som en separat Function App abonnerer på, slik at behandlingskjeden kan fortsette automatisk. Denne arkitekturen understøtter asynkron prosessering og muliggjør høy skalerbarhet ved hjelp av hendelsesdrevne mekanismer i Azure-plattformen. Det er imidlertid viktig å merke seg at ACA per dags dato ikke tilbyr støtte for GPU-akselerasjon, noe som kan være en begrensning ved bruk av ressurskrevende modeller som `whisper-large`.

Ved bruk av Azure Container Instances (ACI) lagres dataene først i en Blob Storage-konto, hvor Function Appen `transkribering-t-bachelor2025` overvåker og lytter etter nye filer. Når en ny JSON-fil oppdages, oppretter denne Function Appen en ny ACI-instans og sender JSON-filen som input til containeren. Resultatet av transkriberingen lagres deretter i en separat container i samme lagringskonto, hvorpå en annen Function App trigges for videre behandling. Denne arkitekturen muliggjør isolert og hendelsesdrevet databehandling, som effektivt kan håndtere store mengder lyddata gjennom en trinnvis prosessering. I motsetning til Azure Container Apps (ACA), støtter ACI GPU-akselerasjon, noe som er fordelaktig for ressurskrevende oppgaver som talegjenkjenning.

3.6.6 Ressurshåndtering i ACI og ACA

En viktig forskjell mellom Azure Container Instances (ACI) og Azure Container Apps (ACA) er hvordan de håndterer ressursallokering. ACI tillater spesifikk og fleksibel tilpasning av CPU og RAM. Dette gir bedre kontroll over ressursforbruket, noe som er nyttig i scenarier hvor ytelse og kostnad må balanseres nøyaktig.

ACA, derimot, benytter forhåndsdefinerte ressursprofiler (presets) for CPU og minne. Dette forenkler skalering og drift, men gir mindre fleksibilitet ved spesialiserte behov. ACA egner seg derfor godt for webapplikasjoner og mikrotjenester med varierende belastning, mens ACI passer bedre for batch-prosesser og arbeidsoppgaver med eksplisitte ressurskrav.

Vi har gjennomført en analyse av ressursbehovet for transkriberingstjenesten, og konkludert med at `nb-whisper-large`-modellen krever minimum 1 CPU og 8 GB RAM for ACI og 4 CPU og 8GB RAM for ACA for å kunne operere stabilt. Dette kravet er nødvendig for å sikre effektiv prosessering av lyddata uten flaskehalser, særlig ved håndtering av lengre eller mer komplekse lydopptak. Tilstrekkelige ressurser er avgjørende for å opprettholde ytelse og redusere ventetid i produksjonsmiljøet.

3.7 Skjema-løsning

Det ble utviklet en enkel og brukervennlig HTML-side for innrapportering av feil, hvor brukeren kan legge inn informasjon om seg selv samt detaljer om

feilmeldingen. Valget av en minimalistisk løsning ble gjort i samråd med oppdragsgiver, som ønsket et lavterskel grensesnitt for sluttbrukerne.

HTML-siden er distribuert som en statisk webside via Azure Storage Account. Brukerens innsendte data håndteres av en Azure Function App, som mottar og prosesserer forespørslene fra skjemaet. Denne løsningen sørger for en kostnadseffektiv og skalerbar arkitektur, samtidig som den ivaretar krav til enkelt vedlikehold og utvidbarhet.

3.8 Telefoniløsning

For telefoni har vi valgt å benytte Azure Communication Service (ACS). ACS tilbyr en skalerbar og sikker plattform for tale- og videoanrop, meldinger og andre kommunikasjonsfunksjoner. Ved å bruke ACS kan vi enkelt integrere telefonifunksjonalitet i applikasjonen uten å måtte bygge infrastrukturen fra bunnen av. Tjenesten støtter også integrasjon med andre Azure-komponenter, noe som gir oss fleksibilitet og mulighet for videreutvikling av kommunikasjonstjenestene i fremtiden [31].

Telefonisystemet benytter såkalte "playbacks" som spilles av etter at brukeren har utført en handling. Dette er til stor hjelp for brukeren, da det veileder gjennom systemet uten at brukeren må ha forkunnskaper om hvordan produktet fungerer — noe som ikke alle brukere nødvendigvis har.

Telefonisystemet benytter også et tastesystem der brukeren får veiledning om tilgjengelige valg, for eksempel tast 1 for å oppgi ansattnummer, tast 2 for å gå direkte til opptakssystemet, og tast 3 for å starte sanntids-transkribering. Hvis for eksempel brukeren har tastet inn sitt ansattnummer, så får brukeren en "playback" som forteller at samtalen blir tatt opp og senere transkribert.

I det nåværende systemet har innmelderen 3 minutter på seg til å si inn feilmeldingen. Dette er et bevisst tiltak for å begrense kostnader knyttet til lange opptak, spesielt i tilfeller der brukeren glemmer å legge på. Under utviklingen av produktet oppdaget vi at uten denne begrensningen kunne uavbrutte samtaler medføre betydelige kostnader.

Når brukeren legger på, trigges en hendelse som fanges opp av en Azure Function-App. Denne funksjonen lagrer informasjon om opptaket, inkludert hvor opptaket kan lastes ned, samt tilleggsdata fra samtalen, som for eksempel ansattnummer. Videre håndteres automatisk prosessering av samtalene av flere Function-Apps i skyplattformen, noe som sikrer en effektiv og automatisert arbeidsflyt.

I tillegg har vi utviklet en web-applikasjon. Under utviklingen opplevde vi at testing av produktet med tradisjonell telefoni medførte betydelige kostnader. Ved å lage en web-applikasjon kunne innringeren i stedet ringe gjennom VoIP, noe som reduserte kostnadene betydelig. Dette gjorde det enklere og mer kostnadseffektivt å teste og videreutvikle telefoniløsningen underveis i prosjektet.

Vi vurderte også å utvikle en mobilapplikasjon, men på grunn av tidspress valgte vi å utsette dette til et senere tidspunkt.

3.9 Transkriberingsløsning

Til transkribering har vi valgt å benytte NB-Whisper, en norsktilpasset variant av OpenAI sin Whisper-modell. Valget ble tatt i samråd med Bodø Kommune, som anbefalte modellen på bakgrunn av dens høye nøyaktighet ved transkribering av norsk tale.

NB-Whisper er en språkmodell som er finjustert fra OpenAI Whisper og spesielt tilpasset det norske språket. Modellen har dokumentert forbedret evne til å forstå ulike norske dialekter, inkludert Bodødialekten, sammenlignet med standardmodellen [32]. Dette gjør NB-Whisper særlig godt egnet for Bodø kommunes arbeidsflyt, hvor det meste av talematerialet er på Bodødialekt eller andre norske dialekter.

Ettersom NB-Whisper kun er tilgjengelig via Hugging Face og ikke gjennom Azure, innebar dette enkelte tekniske utfordringer ved valg av implementasjonsstrategi. Modellen hentes ved hjelp av `pipeline`-modulen fra Hugging Face sitt `transformers`-bibliotek, hvor man oppretter en funksjon for automatisk talegjenkjenning (ASR) og spesifiserer ønsket modell som et argument.

Basert på omfattende testing og kravspesifikasjoner fra oppdragsgiver konkluderte vi med at varianten `NB-Whisper-large-verbatim` er best egnet for vår brukssituasjon. Denne modellen er spesialutviklet for å levere en mer ordrett transkripsjon, i motsetning til den standardiserte `NB-Whisper-large`-modellen som i større grad tolker og omformulerer innholdet. I kontekster hvor presisjon og detaljrikdom er avgjørende – slik som i vårt prosjekt – gir `verbatim`-varianten betydelige fordeler.

Vi har derfor valgt å benytte den store varianten av denne ASR-modellen i de mest kritiske delene av systemet, spesielt i forbindelse med klassifisering av innmeldinger. Dersom transkripsjonen inneholder unøyaktigheter eller mangler, øker sannsynligheten for feilklassifisering. En mer nøyaktig transkripsjon er dermed avgjørende for å sikre høy kvalitet i den videre automatiserte behandlingen av dataene.

Valget av NB-Whisper er dermed både teknisk og organisatorisk fundert, og i tråd med et uttalt ønske fra oppdragsgiver.

3.9.1 NB-Whisper-teknologi

For at NB-Whisper skal kunne transkribere lyd, kreves det en lydfil som inndata, typisk i MP3- eller WAV-format. Lydfilen deles opp i segmenter basert på en angitt chunk size (lengde i sekunder), hvor hvert segment deretter transkriberes individuelt. Ifølge anbefalinger fra NB AI Lab bør chunk size settes til 28 sekunder for å oppnå best mulig ytelse og nøyaktighet [33].

Chunk size: Chunk size refererer til hvordan lydfilen deles opp i segmenter for prosessering. Mindre chunks muliggjør transkribering i sanntid, men på bekostning av kvalitet, ettersom språkmodellen bak Whisper får redusert kontekst og dermed lavere nøyaktighet. For store chunks gir riktignok bedre kontekst og dermed mer presis transkribering, men kan føre til økt bruk av RAM og prosesseringskapasitet. Whisper-modellen er optimalisert for segmenter på rundt 30

sekunder, og NB AI Lab anbefaler derfor en chunk size på 28 sekunder for best mulig balanse mellom ytelse og nøyaktighet.

Mulighet for sanntidstranskribering: For å gjøre sanntids transkribering mulig må chunk størrelsen reduseres til for eksempel ett eller to sekunder, samt strømme lyd direkte, i stedet for å behandle hele opptak i etterkant. Dette kan være brukbart for sanntids brukerstøtte når innmelder melder inn feil i våres sammenheng, men per nå finnes det ingen implementasjon for dette..

whisper.cpp: Et alternativ er å benytte `whisper.cpp`-varianten, som kan gi bedre ytelse i sanntidsscenarier, men som kun støtter WAV-format. Dette gjøres ved å konvertere NB-Whisper-modellen til GGML-format (GPT-Generated Model Language), som brukes i `whisper.cpp`. NB AI Lab har publisert en variant av modellen i GGML-format. Denne tilnærmingen krever imidlertid mer spesialisert kompetanse innen transkriberingsteknologi, samt programmeringsferdigheter i C++ og kjennskap til GGML-biblioteket [33].

Sanntidstranskribering og kompleksitet: På grunn av kompleksiteten ved sanntidstranskribering, og det faktum at sluttresultatet ofte blir bedre ved prosessering i større *chunk*-størrelser, har vi valgt å kun håndtere lydfiler som er tatt opp på forhånd. Disse prosesseres deretter i en Azure Container Instance (ACI) eller Azure Container Apps (ACA). Transkriberingen utføres ved hjelp av NB-Whisper-modellen gjennom Hugging Face sin `Transformers` pipeline-modul i Python, for å holde løsningen så enkel og robust som mulig.

3.9.2 Transkribering: skytjenesteløsning

Transkriberingsprosessen er implementert som en kjede av Azure Function Apps som reagerer på JSON-filer opplastet til en container i en Azure Storage Account.

Prosessen starter med en telefonsamtale. Når innmelderen legger på, blir lydopptaket tilgjengelig via en URL. Denne URL-en, sammen med annen metadata om samtalen, sendes som en hendelse via Azure Event Grid. Den første Function App-en reagerer på denne hendelsen, henter inn all relevant informasjon, og pakker dette sammen i en JSON-fil.

Det er satt opp to alternative løsninger for videre prosessering: enten ved bruk av Azure Container Instances (ACI) eller Azure Container Apps (ACA).

ACI-løsning: JSON-filen lagres i en container i Azure Storage, noe som utløser en blob-hendelse. Denne hendelsen fanges opp av en annen Function App, som overvåker containeren og iverksetter videre prosessering. Når en ny JSON-fil oppdages, opprettes en Azure Container Instance som kjører NB-Whisper-modellen for transkribering. Resultatet lagres som en ny JSON-fil med den transkriberte teksten samt tilhørende metadata, for eksempel ansattnummer. Denne arkitekturen gir en skalerbar, parallell og fleksibel løsning for behandling og transkribering av lydopptak i skyen, og støtter en hendelsesdrevet arbeidsflyt med tydelig ansvarfordeling. For ACI betales det kun for selve kjøretiden til containerinstansen.

ACA-løsning: JSON-dataen (som strukturert HTTP-body, ikke som fil) sendes direkte fra telefonsystemet til en Azure Container App som fungerer som en webapplikasjon. Applikasjonen eksponerer for eksempel ruten `https://transkribering-aca.mangomushroom-` hvor transkriberingen utføres umiddelbart. Resultatet returneres som et HTTP-respons og lagres i tillegg i en blob-container. Denne løsningen gir en enklere og potensielt rimeligere infrastruktur enn ACI, men dette må evalueres gjennom praktisk testing. For ACA betales det for både oppstartstid og de operasjonene som utføres i containeren. Det vil si at kostnaden dekker tiden containeren er aktiv, inkludert oppstart. Når containeren ikke er i bruk, skaleres den ned til null, noe som innebærer at det ikke påløper kostnader i denne perioden. Grunnen til at det påløper kostnader for oppstartstid, er at det krever dataressurser å starte opp og sette i verk ACA-containeren. Ulempen med den nåværende ACA-implementasjonen er at den mangler støtte for parallell prosessering. Dette medfører at flere samtidige HTTP-forespørsler kan føre til krasj i systemet. Dette står i kontrast til ACI-implementasjonen, som håndterer samtidige forespørsler bedre ved at det opprettes en ny ACI-instans for hver transkriberingsprosess. Dette gir isolasjon mellom prosessene og gjør systemet mer robust og skalerbart ved høy belastning.

Valg av arkitektur overlates til oppdragsgiver, basert på krav til ytelse, kostnad og kompleksitet.

Utviklings problemer: Å utvikle transkriberingsløsningen i en skytjeneste viste seg å være betydelig mer utfordrende enn å gjøre det lokalt på en datamaskin. Vi startet med å eksperimentere med Azure Function Apps, og det fungerte tilfredsstillende lokalt – en Function App trigget av opplastede blobfiler kunne utføre transkribering. Imidlertid oppstod problemer da vi forsøkte å kjøre den samme løsningen i Azure. Etter omfattende feilsøking fant vi ut at modellen "Transformer pipeline" var for stor til å kjøre i en vanlig Function App i skyen.

Som et alternativ vurderte vi å bruke containere. Vi forsøkte først å kjøre Function Appen i en containerisert versjon, men også dette mislyktes. Løsningen kom da vi tok i bruk Azure Container Instances (ACI), som viste seg å støtte modellen og tillot vellykket transkribering. Senere i prosjektet fant vi også ut at Azure Container Apps (ACA) kunne benyttes som et alternativ, enten gjennom såkalte jobs eller som en webapplikasjon som kontinuerlig lytter til HTTP-forespørsler, og automatisk skalerer ned til null når den ikke er i bruk.

En utfordring med bruk av Azure Container Instances (ACI) er at kostnaden baseres på kjøretid, i motsetning til Azure Container Apps (ACA), som kan skaleres til null og prises mer etter antall operasjoner. I vår løsning opprettes det en ny ACI-instans for hver transkriberingsprosess. Dette fører til mange separate forekomster i Azure-portalen. Dette gjør systemet mer uoversiktlig og vanskeligere å navigere for administratorer og utviklere.

For å møte denne utfordringen besluttet vi å utvikle en mekanisme som automatisk sletter inaktive ACI-instanser etter et visst tidspunkt – eksempelvis hver dag ved midnatt. Fordelen med denne tilnærmingen er at det bidrar til en ryddigere portal, og gjør det enklere å få oversikt over aktive prosesser. En potensiell ulempe er at man kan miste historisk informasjon, som kunne vært

nyttig for feilsøking og debugging.

For å bøte på dette kan man lage et system som tar vare på relevante logger og annen informasjon fra containerne i en Azure Blob Storage-konto, før instansene termineres. Dette gjør det mulig å bevare viktig driftsinformasjon, samtidig som systemet holdes rent og oversiktlig.

3.10 Lagring av persondata

I starten vurderte vi å bruke en database for lagring av persondata, men underveis i prosjektet fant vi ut at Azure Table Storage var tilstrekkelig for våre behov. Ved et senere tidspunkt kan oppdragsgiver selv vurdere en database-basert løsning dersom behovet endrer seg.

Lagring av persondata håndteres av en Azure Function App som reagerer på transkriberingsfiler i JSON-format, som legges inn i en Blob Storage-container. Funksjonen benytter en stor språkmodell (LLM) til å hente ut relevant informasjon både fra selve transkriberingen og tilhørende metadata, som for eksempel ansattnummer. Den uttrukne informasjonen lagres deretter i Table Storage.

4 Resultater

4.1 Vitenskapelige resultater

I denne oppgaven ønsker vi å se på egenskapene til feilmeldingssystemet vi har utviklet, og måle hvor godt det utfører oppgavene sine. Tid brukt fra innmelder leverer feilmelding til systemet og fram til en klassifisering er gitt, er en kvantifiserbar verdi, som vi kan måle og sammenligne med kravene Bodø kommune setter til dagens leverandør av kundebehandling. Hvor korrekt klassifiseringene er, vil derimot være kvalitative vurderinger, og vi vil diskutere verdiene gitt av systemet ved klassifisering, og sammenligne disse med eldre klassifiseringer for å argumentere verdien av produktet. Det gjelder også transkriberingen av tale-til-tekst som blir utført ved innringing av feilmelding, og her vil vi vurdere hvor korrekt transkribert tekst er i forhold til intensjonen til innringer, og om RAG klarer å gi vurdering basert på den transkriberte teksten.

4.1.1 Transkriberingsresultater

For å teste ut transkriberingen har vi benyttet nb-whisper-verbatim med anbefalt innstilling fra NB AI Lab med chunk size 28 og num_beams på 5 og dette kjøres i skytjenesten azure som en ACI med bare 1 CPU, gpu er ikke tatt i bruk. Vi har brukt disse innmeldingene, hvor navn er anonymisert i rapporten:

1. ***Har ikke internett. Kan det undersøkes og fikses. Hilsen [navn], Helsesykepleier, Alstad Barneskole, Helsetunet.***
2. ***Hei, jeg har problemer med CGM. Får ikke noen feilmelding, appen bare stopper opp. Jeg har prøvd å slå av og på pcen, men det funker fortsatt ikke.***
3. ***Hei, får ikke til å skrive ut avansert journalutskrift fra CGM. Jeg kan skrive ut dagens journal. Det gjelder alle pasienter. Jeg bruker dette sjeldent derfor er jeg usikker når det fungerte sist. Startet Citrix Workspace på nytt, avsluttet Citrix sesjoner, samme problemet. Lokasjon: Kommunelegen i Rødøy.***
4. ***Jeg får ikke koblet meg opp på kommuneserveren og får ikke logget meg på LMP. Dette gjelder hele avdelingen på hjemmetjenesten Østbyen.***
5. ***Det er ikke nett på lokasjonen min, Hålogalandsgata 91. Både kabel og wifi er nede. Alle på lokasjonen er påvirket. Jeg får ikke logget på pcen uten nett.***
6. ***Legekontor får ikke nett til å virke. Alderstun omsorgssenter Skolebakken 110 8186 Tjongsfjord IPVPN. Helse og omsorg 50. Begge to kabel og nett er borte Bodø kommune. Fredag fungerte kabel men ikke trådløss. feil: enhet er konfigurert riktig, men DNS svarer ikke.***
7. ***Lokasjon: legetjenesten. Kan ikke koble til nettverk, det vises ingen feilmelding, det bare snurrer. Nettverket er BKAD local og wifi. Det er ca. 10 brukere som er påvirket. Nettet har***

fungert dårlig hele dagen, var problemet med å komme inn på CGM, osv. men nå nettverk er borte i det hele tatt.

8. ***Bodø legevakt Plassmyrveien 13. Mangler nettverk.***
9. ***Hei, feil med LMP, e-Rom og trådløs nett ved Alderstun omsorgssenter. Trådløs nett DEV er nede. De pasienter med KOMP og nettbrett får ikke logget seg på.***

Vi har forsøkt å innmelde disse gjennom vårt telefonisystem, og deretter til transkriberingssystemet, hvor vi fikk disse som resultat i rekkefølge:

1. ***ja han skulle sende inn en feilmelding jeg har ikke internett kan det undersøkes og fikses hilsen [navn] helsesykepleier alstad barneskole helsetjeneste***
    - **Transkriberings tid: 253 s**
2. ***ja hei jeg har problemer med cgn får ikke noen feilmelding appen bare stopper opp jeg har prøvd å slå av og på pc-en men det funka fortsatt ikke***
    - **Transkriberings tid: 522 s**
3. ***hei får ikke til å skrive ut annonserte journal journalutskrift fra cgm jeg kan skrive uten dagens journal det gjelder alle pasienter jeg bruker dette sjelden derfor er jeg usikker når det fungerte sist startet et sitrix first space på nytt avsluttet sitrix-sesjoner samme problemet lokasjon kommunelegen i rødøy***
    - **Transkriberings tid: 545 s**
4. ***hei jeg får ikke koble meg opp på kommuneserveren hun får ikke logge meg på all amp dette gjelder hele avdelingen på hjemmetjenesten østbyen***
    - **Transkriberings tid: 330 s**
5. ***det er ikke nytt på lokasjonen min håla hålogalandsgate 91 både kabel og wifi er nede alle på lokasjonen er påvirka får ikke logge på pc-en uten nett***
    - **Transkriberings tid: 307 s**
6. ***legekontor får ikke nett til å virke aldersstum omsorgskjente skolebakken 110 åtte 186 tongsfjord ip wpm helse og omsorg 50 begge to kabler og nett er borte fra bodø kommune fredag fungerte kabelen og han ikke tråler oss feil enhets han konfire konfigurerte riktig men dns svarer ikke***
    - **Transkriberings tid: 670 s**
7. ***lokasjon legetjenesten kan ikke koble til nettverket det vises ingen feilmelding det bare snurrer nettverket er bcads lokal og wifi det er lka ti brukere som harvirker nettet har fungert dårlig hele dagene jeg har problemer med å komme inn på cgm osv men nå er nettverket borte i det hele tatt***

- **Transkriberings tid: 902 s**

8. ***bodø legevakts plassmyhrveien 13 mangler nettverk***

   - **Transkriberings tid: 217 s**

9. ***feil med allmp erom og trådløs nett ved olderstund omsorgssenter trådløs nett div er nede de pasienter med komp og knaptbrett får ikke logge seg på***

   - **Transkriberings tid: 266 s**

Ut fra resultatene kan vi se at modellen har klart å fange opp det meste av innholdet. Det forekommer enkelte småfeil, men generelt er transkripsjonene forståelige. Merk at vi har benyttet `nb-whisper-verbatim`, som er en modell designet for å fange opp alt brukeren sier – inkludert fyllord, pauser og ustrukturert tale. Valget av denne modellen ble gjort med hensikt, da den gir et mer detaljert bilde av det som blir sagt. Dette gir et godt utgangspunkt for videre behandling, for eksempel ved bruk av en LLM (Large Language Model) som kan analysere transkripsjonene og hente ut relevant informasjon automatisk.

Vi fikk testet noen av eksemplene ved å kjøre nb-whisper-large i en ACA-instans med 4 vCPU og 8 GB RAM.

1. ***Hei. Får ikke til å skrive ut avansert journalutskrift fra CGM. Jeg kan skrive ut dagens journal. Det gjelder alle pasienter. Jeg bruker dette sjelden, derfor er jeg usikå når det fungerte sist. Startet Seatricks Workspace på nytt? Avsluttet Seatricks-sesjoner med samme program? Lokasjonen kommunelegen i Rødøy?***

   - **Transkriberings tid: 162 s**

2. ***Jeg får ikke kobla meg opp på kommuneserveren og får ikke logga meg på LMP. Dette gjelder hele avdelingen på Hjemmetjenesten Østbyen.***

   - **Transkriberings tid: 76 s**

3. ***Det er ikke nett på lokasjonen på Lågalandsgata 91. Både kabel og wifi er nede. Alle på lokasjonen er påvirka. Jeg får ikke logget på PC-en utenat.***

   - **Transkriberings tid: 293 s**

nb-whisper-large viste seg å gjenkjenne forkortelser som LMP og CGM bedre enn nb-whisper-large-verbatim. Likevel var det fortsatt enkelte ord, som Citrix, som ble feiltolket – i dette tilfellet transkribert som Seatricks. Hålogalandsgata 91 ble Lågalandsgata 91, dette kan være at taleren var litt uklar.

4.1.2 RAG Resultater

Vi har testet ut RAG på transkriberingen fra "Jeg får ikke koblet meg opp på kommuneserveren og får ikke logget meg på LMP. Dette gjelder hele avdelingen på hjemmetjesten Østbyen" eksempelet.

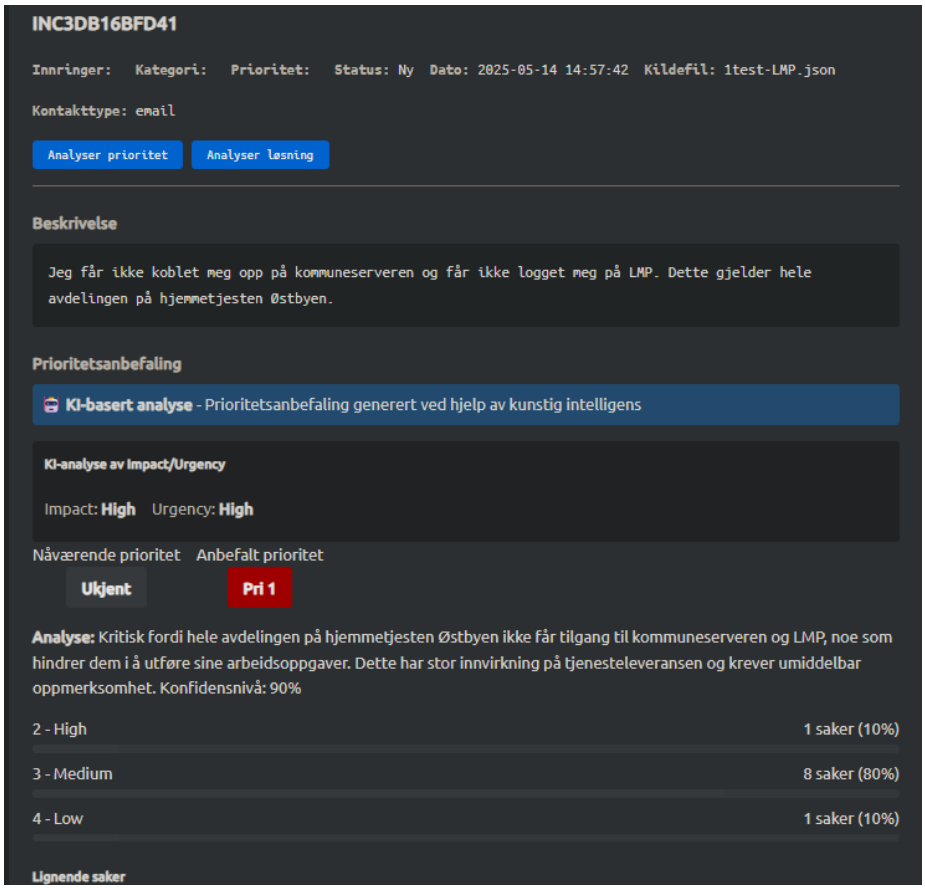

Figure 1: Eksempel på analysert feilmelding

Bodø kommunes krav responstid til innmeldte feilmeldinger baserer seg på klassifiseringen av feilen, hvor alvorlige feil krever en lav responstid. Vi ønsker at systemet behandler sakene raskt nok til at det er mulig å respondere på feilen (sette i gang tiltak, feilsøke, osv.) innenfor gitt responstid. På en kritisk feil er kravet at respons på feilen skal være initiert innen 15 minutter.

Like viktig som at systemet behandler klassifisering hurtig, er at klassifisering som blir gitt er korrekt. Vår testing av klassifisering har vært en blanding av egenkomponerte feilmeldinger og gamle feilmeldinger. De egenkomponerte feilmeldingene lar oss teste spesifikke scenarioer som kan være vanskelige for KI å vurdere, slik at vi kan finne hvor begrensningene til systemet ligger. De gamle feilmeldingene gir oss en mulighet til å sammenligne direkte med dagens system og undersøke om resultatene blir like eller ulike.

Table 2: Hendelser som Bodø kommune klassifiserer som kritiske

Beskrivelse av feilmelding	**Klassifisering av RAG**
Bruker får ikke logget på tjenesten LMP som brukes i hjemmetjenesten	1 - kritisk

Problem med tjenesten CGM	3 - medium
Problem med å skrive ut avansert journal fra CGM	3 - medium
Nettverk er nede på hele innmeldte lokasjonen	1 - kritisk
Nettverk er nede på legesenteret	2 - høy
Nettverk er nede på legetjenesten med flere systemer påvirket	1 - kritisk
Bodø legevakt mangler nettverk	2 - høy

Feilmeldingssystemet har som oppgave å finne eldre saker med liknende feilsymptomer eller som berører samme system, og vi testet hvor relevante disse hendelsene var til den feilmeldingen som var sendt inn. I de fleste tilfeller henviste KI til saker som var relevante for feilmeldingen. Se vedlegg for eksempel på feilmelding med tilhørende liknende historiske hendelser.

Figure 2: Eksempel på lignende hendelser

4.2 Ingeniørfaglige resultater

Løsningen vår er en blanding av kode i Python og deployment av tjenester i Azure. Vi har brukt Azure-tjenester til å gjøre store oppgaver som ville vært vanskelige å lage på egenhånd, og vi har også brukt Azure til å hoste den ferdige løsningen. Det meste av logikken i produktet styres av Python-kode som kjøres i Azure Function App-er. Dette er applikasjoner som har innebygde triggere som kan kjøre kode dersom en bestemt hendelse oppstår, som for eksempel når en ansatt ringer nummeret til tjenesten. Fordelen med å bruke en løsning som ligger helt i skyen, er at det er kort vei fra test til produksjon.

I henhold til kravspesifikasjonene gitt i begynnelsen av prosjektet, er de funksjonelle kravene til produktet møtt helt eller delvis. Den første rekken med krav vi satte til produktet handlet om funksjonalitet knyttet til innmeldingen av feilmeldingen. Det er mulig å ringe inn og melde fra informasjon knyttet til feilen som skal rapporteres. Produktet har et dedikert telefonnummer som man kan ringe, og produktet vårt svarer innringer automatisk. Deretter blir det spilt av en lydfil som gir instruksjoner om hvilken informasjon som kreves av innringer for

å melde fra om feil. Samtalen blir tatt opp automatisk og lagret i skyen. Lydfilen transkriberes fra tale til tekst med NB-Whisper og lagres som en tekstfil i Azure.

Ett av kravene vi satte ved begynnelsen av prosjektet, var at systemet skulle være automatisk fra innmeldt feilmelding til ferdig klassifisering, men vi har funnet at det ble vanskelig å møte dette kravet helt. Slik produktet er nå, er alt automatisk bortsett fra genereringen av klassifisering og forslag til løsning. Disse krever brukerinput, men kun for å sette i gang prosessen. Personen som er ansvarlig for feilretting vil få opp hendelsen i en dedikert web-applikasjon hvor hendelsene dukker opp automatisk etter ferdig transkribering. Her må personen velge hendelsen og trykke på en knapp for å sette i gang prosessen for klassifisering, og en annen knapp for å finne lignende hendelser. Det er altså ikke mye som kreves av behandler annet enn å trykke på to knapper, men det betyr at systemet ikke er helautomatisk.

Table 3: Funksjonelle krav

Krav	**Status**
Feilmelding via telefon	Det er mulig å ringe inn som ansatt via et spesifikt nummer.
Feilmelding via skjema	Produktet har et skjema med relevante felter som kan brukes av ansatte til å melde inn feilmeldinger. Skjemaet er ikke på Bodø kommunes nettsider siden produktet ikke er i bruk, men det skal være mulig å gjøre dette senere.
Transkribering av tale til tekst	Etter at ansatt har meldt inn feilmeldingen på telefon, blir lydfilen transkribert ved bruk av NB-Whisper.
Lagring av talemelding i Azure	Talemeldingen og den transkriberte versjonen av talemeldingen blir lagret i en Azure Storage Account
Feilklassifisering ved bruk av RAG	Innmeldt feilmelding blir klassifisert automatisk av RAG i henhold til krav spesifisert av Bodø kommune
Bruk av RAG til å finne lignende hendelser i eldre saker	Ved behandling av feilmeldingen vil produktet lete i gamle hendelser for å finne liknende saker. Disse listes opp sammen med løsningsnotat, slik at feilretter kan se hva som ble gjort sist problemet oppsto
Forslag til feilårsak fra KI	Ved behandling av feilmelding vil produktet gi et forslag til hvordan feilen kan løses. Som oftest vil dette være en oppsummering av løsningene som ble funnet i søket av lignende gamle hendelser
Automatisert gjennomgang fra innmelding til ferdig klassifisering	Produktet gjennomfører deler av prosessen automatisk.

Table 4: Ikke-funksjonelle krav

Krav	**Status**
Skytjeneste	Vi benytter oss av skytjenesten Azure til å hoste og kjøre produktet
Personvern	Systemet forsøker å følge personopplysningsloven, men mangler enkelte deler for å være helt etterrettelig. Se videre diskusjon nedenfor
Oppetid	Produktet vil ha en stabil oppetid fra 08-16 såframt Azure ikke er nede.
Samtidig bruk	Vi har testet at produktet kan brukes av minst to personer samtidig

Ettersom produktet bruker tjenester i Azure, er vi avhengig av Azures oppetid for å bestemme produktets oppetid. Ifølge Microsofts SLA for de tjenestene produktet benytter, vil en prosent av oppetid på under 99.9% gi oss krav på service credit på 10% av bruk og dersom den er under 99% vil det gi oss krav på service credit på 25% av bruk [34]. Vi kan derfor regne med at oppetid for produktet vårt vil være tilnærmet 100%. Dette gjelder ikke dersom det er planlagte oppdateringer av Azure, hvor det potensielt kan være lengre nedetid. Kravet vi satte oss ved starten av prosjektet var at systemet skal være oppe i løpet av vanlig arbeidstid, fra 08-16.

For at produktet skal være tillatt å bruke i aktiv drift, er det nødt til å oppfylle kravene til personopplysningsloven. Loven består av EUs personvernsforordning, General Data Protection Regulation (GDPR), i tillegg til nasjonale lover. GDPR bestemmer hvordan personlig data skal lagres og håndteres, og det er hovedsaklig GDPR som er relevant for vårt produkt. Mer detaljert, er det nødvendig å lagre data trygt, få tillatelse av vedkommende til lagringen, og i tillegg kunne hente fram data ved etterspørsel, og også slette data etter en gitt periode. Ettersom vi bruker Microsoft Azure som skyløsning, er vi sikret at data blir lagret trygt og vi får muligheter til å automatisere sletting av data etter en fastsatt tidsperiode. Det er mulig å hente ut brukerdata fra Azure, men vi har ingen systemer som gjør dette automatisk. Vi har laget systemer for å få samtykke til lagring av data ved innringing, ved at innringer får en beskjed om at samtalen blir tatt opp og at de kan trykke på 0 på tastaturet for å samtykke til innspillingen.

GDPR krever også at det utføres en Data Protection Impact Assessment (DPIA) for å kartlegge risiko for personvernsbrudd. Vi hadde planlagt å gjennomføre denne, men fant at det ble en for stor oppgave å få det gjennomført til dette prosjektet.

4.3 Administrative resultater

Vi var noe optimistiske til hvordan utviklingen kom til å gå ved begynnelsen av prosjektet, og undervurderte hvor sammensatt enkelte av problemene var. Vi brukte lengre tid enn forventet på å sette opp RAG-implementering og telefoni-integrering. I tillegg valgte vi å bruke Azure Container Instances til å kjøre koden til transkriberingen, og dette var ikke planlagt i forkant av prosjektstart. Vi benyttet oss ikke av databaser og dette punktet er fjernet fra oversikten.

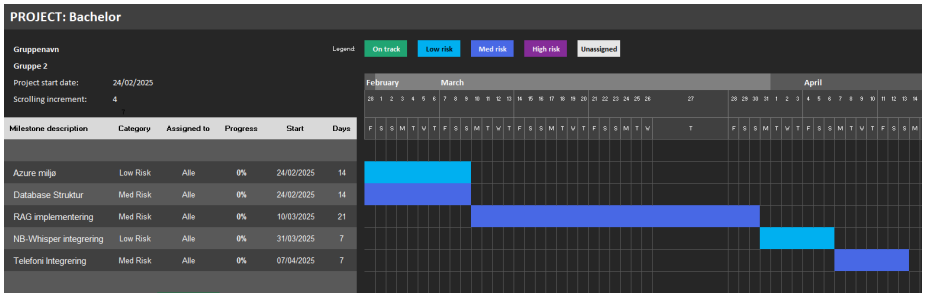

Figure 3: Gantt-diagram ved prosjektetsstart

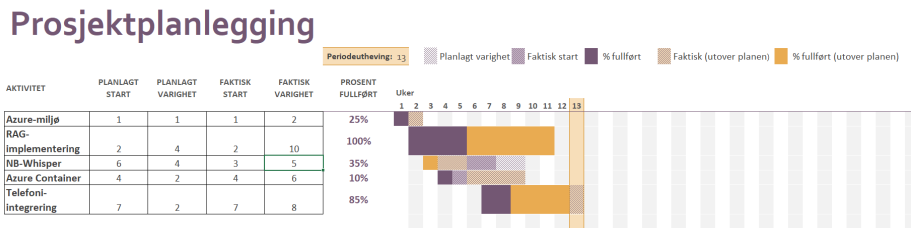

Figure 4: Gantt-diagram ved prosjektslutt

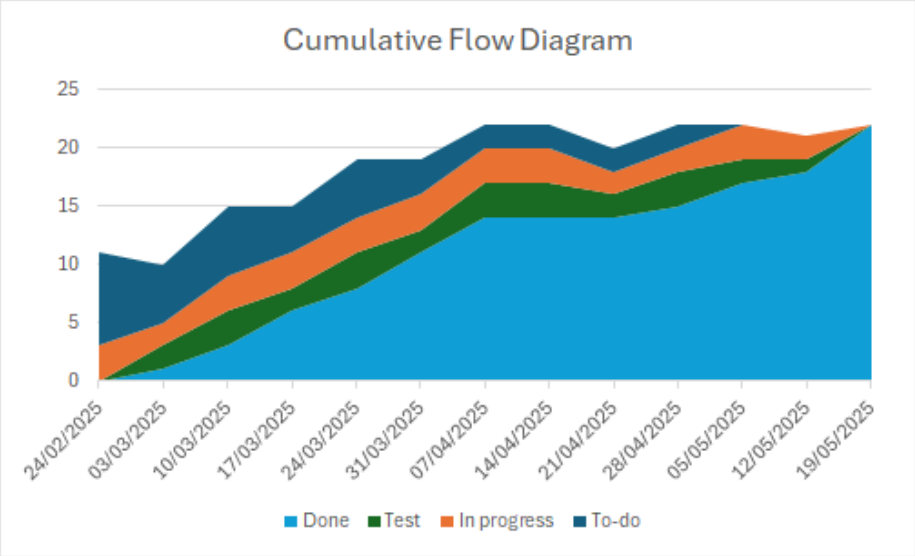

Figure 5: Cumulative Flow Diagram

5 Diskusjon

Formålet med prosjektet har vært å skape et produkt som leverer en bedre brukeropplevelse ved å behandle innmeldte feilmeldinger raskere, ved å være mer konsekvente i vurderingene, og ved å lage et enklere system for ansatte å melde inn feil. I tillegg hadde vi et mål om at produktet skulle være billigere i bruk kontra dagens system. De påfølgende paragrafene vil gå nærmere inn på hvert av disse punktene og diskutere om vi har klart å oppfylle kravene.

5.1 Behandlingstid og krav til responstid ved feilmeldinger

Produktet leverer ferdig klassifiserte feilmeldinger på kort tid. Det er viktig at systemet håndterer feilmeldingene som blir innmeldt hurtig, slik at eventuelle feil kan utbedres uten lengre ventetid. I dagens Service Level Agreement (SLA), har Bodø kommune satt krav til hvor lang tid driftspartner har på å svare på feilmeldingen, og hvor lang tid de har på å starte feilrettingen. Tidsrammen baserer seg på alvorlighetsgraden av feilen og er kortere ved alvorlige feil. Ettersom vi ikke vil være ansvarlige for å rette opp feilen, vil fokuset være på tiden brukt fra ansatt i kommunen har sendt inn en feilmelding, til feilmeldingen er ferdig behandlet.

5.2 Kostnader

Kostnader ved å drifte systemet. Vi har, som nevnt tidligere, deployet produktet i Azure, og mesteparten av driftskostnadene går på å betale for bruk av diverse Microsoftprodukter i Azure. Den totale månedskostnaden for Mai under utviklingen har ligget på ca. 2436 kroner, hvor bruk av telefon og kunstig intelligens har vært de største kostnadene. Mai er valgt som den mest representative måneden for å illustrere produktets drift, da den gir et realistisk bilde av

kostnadene ved å holde systemet i gang etter den siste utviklingsfasen. Noen av disse kostnadene vil øke når systemet settes over i produksjon og bruken økes, men flere av kostnadene er faste priser som ikke vil påvirkes av volum. Se tabellen for nærmere kostnadsoversikt. Vi vil påstå at produktet er mye billigere i drift enn dersom man skulle benyttet en servicedesk eller lignende tjeneste til å utføre oppgaven.

Table 5: Kostnader knyttet til bruk av produktet fra 01.05 til 20.05

Tjeneste	**Kostnad**
Azure AI Search	535 kr
Azure Communication Services	1634 kr
Azure Storage Container	193 kr
Log Analytics	61 kr
Totalt (inkludert mindre beløp)	2436 kr

5.3 Transkriberingsnøyaktighet

Transkribering av tale-til-tekst har den ulempen at det ikke er mulig å få en klargjøring av innringer om hva som ble sagt. Vi oppdaget tidlig at det kom til å bli et problem å få korrekt transkribering av tall. Blant annet dersom innringer listet opp ansattnummeret sitt var det vanskelig å være sikker på at transkribering var korrekt. For eksempel så vi at "én" kunne bli til "i" og "to, fem" ble til "tomme". Ansattnummer har mye nyttig informasjon om innringer, blant annet kan man finne hvor den ansatte jobber og relevant kontaktinformasjon. Ettersom tallene kunne gi mye støy og feil input, valgte vi å bytte til en løsning hvor innringer kan taste inn ansattnummeret via telefonens tastatur, slik at man er sikret at det ikke er feil i transkriberingen. Et slik løsning er ikke feilfri og innringer kan for eksempel trykke feil nummer.

5.3.1 nb-whisper-large-verbatim resultater

Basert på resultatene fra nb-whisper-large-verbatim så ser man at det har vanskeligheter for å tolke navn på tjenester. Tjenestene har gjerne idiosynkratiske navn eller er forkortelser, og er derfor ikke normale ord som man støter på ofte. Vi ser for eksempel at tjenesten LMP blir transkribert som allmp og Citrix Workspace blir sitrix first space. I tillegg ser vi at stedsnavn også blir feiltolket, for eksempel blir Alderstun til olderstund og aldersstum. Igjen er nok dette fordi det ikke er ofte brukte ord, og NB-Whisper mest sannsynlig ikke er trent på å gjenkjenne dem.

Utenom problemene nevnt ovenfor, er transkriberingen nesten helt lik innlest melding. Det er enkelte ord som blir feil, men meningen med setningen er som som oftest bevart. Med tanke på bruk av meldingene til klassifisering er det viktigste at NB-Whisper korrekt transkriberer ord som beskriver omfang av feil, hvilke systemer eller tjenester som er berørt, eller hvor mange som er påvirket.

Fra testene som er gjort så ser man et eksempel på feiltranskribering der "nettbrett" ble tolket som "knaptbrett". Dette kan skyldes at modellen ble kjørt

i verbatim-modus, hvor kommandoen `--condition_on_previous_text=False` gjør at modellen i mindre grad benytter tidligere tokens til å tolke konteksten. Dette reduserer modellens evne til å forstå sammenhengen i tale og øker sannsynligheten for feil [35]. Dersom transkripsjonen videre bearbeides av en stor språkmodell (LLM), vil denne som regel kunne bruke konteksten til å korrigere feilen og forstå at det mest sannsynlige ordet er "nettbrett". Det kan naturligvis oppstå variasjoner i transkriberingen, blant annet fordi talen kan være uklar. Tilfeldige forhold som bakgrunnsstøy kan også påvirke nøyaktigheten.

5.3.2 nb-whisper-large resultater

Basert på resultatene fra nb-whisper-large, ser vi at denne modellen i større grad klarer å tolke meningsinnholdet i det som blir sagt. Den benytter også skilletegn, noe som forbedrer lesbarheten og gjør transkripsjonene lettere å forstå. Dette gjør modellen godt egnet i situasjoner hvor forståelighet er viktig.

I ett av testeksemplene ble "Hålogalandsgata 91" transkribert som "Lågalandsgata 91". Dette kan skyldes at uttalen i lydopptaket var noe uklar, og viser hvordan modellen kan ha utfordringer med å gjenkjenne egennavn eller lokale stedsnavn når taleren ikke artikulerer tydelig. Slike feil kan ha praktiske konsekvenser dersom adressen skal brukes videre i saksbehandling eller varsling.

På den andre siden kan dette gå på bekostning av detaljer. Sammenlignet med nb-whisper-large-verbatim, som forsøker å gjengi talen mer ordrett, kan nb-whisper-large risikere å utelate eller endre enkelte språklige nyanser som kunne vært nyttige for videre analyse, for eksempel i en språklig, psykologisk eller juridisk kontekst.

5.4 Klassifisering

Til slutt var ett av målene at systemet skulle produsere mer konsekvente feilmeldinger, hvor den ferdige klassifiseringen er i samsvar med kravene til Bodø kommune. I dagens system opplever kommunen at feilmeldinger ofte blir satt til en lavere prioritet enn ønsket. Det betyr at feil som krever hurtig behandling ikke får tilstrekkelig oppmerksomhet og ressurser, noe som kan føre til nedetid for kritiske tjenester.

Gjennom testene vi har gjennomført, har vi funnet at RAG klarer å gi klassifiseringer og henviser til kravene i kritikalitetsmatrisen, men at den ved noen tilfeller kan vise til feil krav, og dermed gi ukorrekt klassifisering. I de tilfellene som gir feil vurdering, ser vi at feilmeldingen som er sendt inn har dårlig informasjon om problemet, eller at kritikalitetsmatrisen mangler relevant beskrivelse av tjenesten. RAG har derfor få holdepunkter når den skal vurdere klassifiseringen. Det vil være vanskelig å forsikre seg at hver innsendte feilmelding inneholder alt av relevant informasjon, og det vil i noen tilfeller derfor være nødvendig med oppfølging av innmelder slik at man kan hente inn mer informasjon om feilen. En mulig løsning på dette problemet er å markere hendelser hvor RAG gir lavt samsvar mellom feilmelding og kritikalitetsmatrise, slik at feilretter kan sjekke etter mulige feil i klassifiseringen.

Et av ønskene fra oppdragsgiver var å sammenligne innsendt feilmelding med tidligere hendelser som var lik den nye, men vi oppdaget at det var problemer

med manglende informasjon knyttet til gamle hendelser. Bodø kommune ga oss en samling av innmeldte hendelser fra 2024 for å bruke som grunnlag til dette formålet. Hendelsene er registrert med en kort beskrivelse av feilen og et kort notat når saken lukkes. Disse meldingene inneholder i flere tilfeller lite eller ingen informasjon om hva som ble gjort for å løse saken. Det betyr at systemet klarer å finne lignende hendelser, men klarer ikke gi noen nyttig informasjon om hva som forårsaket feilen eller hvordan den kan løses. Dette er ikke en feil som kan lastes systemet, men som peker på at det har vært dårlige rutiner på dokumentering av feil tidligere. Vi har tatt opp problemstillingen med oppdragsgiver, som igjen har formidlet problemet videre til selskapet som drifter servicedesken for kommunen.

Oppdragsgiver har vært fornøyd med produktet vi har utviklet, og resultatene produktet har levert. Selv om vi skulle ønsket at produktet var nærmere en ferdig tilstand hvor det kunne blitt brukt aktivt i kommunens feilklassifisering, har resultatene vi har produsert, tydelig vist at det er mye potensiale i en slik løsning. Oppdragsgiver kan videreutvikle produktet senere og få det til et nivå som gjør det mulig å bruke det til aktiv innsamling av feilmeldinger. Oppdragsgiver kan også vise til arbeidet vi har gjort som et eksempel på hva som er mulig når kommunen er i samtaler med IKT-leverandører om tjenester de ønsker å kjøpe. Det har blitt nevnt som en av fordelene med utviklingen vi har synliggjort.

5.5 Juss og etikk

I Norge har vi i dag generelle teknologiregler i stedet for egne regler for hver bransje og hvert produkt. Når man bruker kunstig intelligens må vi derfor følge generelle regler som gjelder for flere områder samtidig. Ett enkelt produkt kan være regulert av flere forskjellige lover. Dette betyr at virksomheten som tar i bruk KI må forstå hvilke regler som gjelder i dag. I tillegg må den følge med på nye regler som kommer, både fra EU og fra norske myndigheter. Når kunstig intelligens bistår i å utføre en arbeidsoppgave i virksomheten, må den følge de samme reglene som ellers gjelder for virksomheten. Lover og regler som gjelder når oppgaven utføres manuelt gjelder fortsatt. Vi må bare tilpasse hvordan vi bruker reglene når den nye teknologien innføres.

Vår vurdering er at KI-system som er utviklet i bacheloroppgaven ikke i faller inn under høyrisiko KI. Vår vurdering er at vår KI-system kan klassifiseres som et anbefalings- og kvalitetssystem. Det forekommer personinformasjon i grunnlaget som brukes. Vi vil anbefale at det gjøres en særskilt vurdering på hvordan denne personinformasjonen kan fjernes slik at løsning ikke kommer i konflikt med personvernet. Den oppgaven som vi ferdigstiller, er en foreløpig versjon av et KI-system som kan utvikles videre av Bodø kommune. Hvis Bodø kommune velger å ta løsningen i bruk i sin daglige virksomhet, mener vi at det bør gjøres en grundig gjennomgang, og at det etableres en behandlingsprotokoll som dokumenterer hvilke data systemet bruker i sin KI-behandling. Da må det dokumenteres at løsningen er i samsvar med KI-forordningen. Selv om KI-forordningen ikke er innført i Norge så vil det være riktig å sikre allerede nå at løsningen tilfredsstiller kravene i forordningen.

5.6 Generell diskusjon

Utviklingen av produktet er en del av en større trend de siste årene, hvor den hurtige forbedringen av kvalitet på output fra KI har ført til økt søk etter områder hvor teknologien kan anvendes. Digitaliserings- og forvaltningsminister Karianne Tung ønsket i april 2024 at 80 prosent av offentlig sektor benyttet seg av KI innen 2025 [36].

En av fallgruvene som kan oppstå ved jakten etter nye bruksområder for ny teknologi, er at det blir brukt ukritisk uten tanke på nytteverdien eller kostnaden knyttet til bruken. De siste årene har nytenkende teknologier som Web3, blockchain og IoT dukket opp og skapt spennende muligheter for nyvinning, men teknologiene kan også representere potensielle tapsprosjekt dersom teknologibruken ikke skaper profitt i form av økte inntekter eller økt produktivitet. Folkefinansieringsplattformen Kickstarter annonserte i 2021 at plattformen skulle flyttes fra en tradisjonell nettside til en Web3-basert nettside [37]. Prosjektet ble raskt møtt med negative reaksjoner fra både kunder og partnere, og Kickstarter ble til slutt nødt til å sette prosjektet på pause [38].

På samme måte som Kickstarter opplevde negativ omtale rundt sin omstilling til Web3, er mulig at det kan komme negative reaksjoner fra ansatte i Bodø kommune til feilmeldingssystemet vi har utviklet. Systemet fjerner alt av mellommenneskelig kontakt, og det kan være at brukere vil reagere negativt på at det ikke lenger er en person de kan forklare problemet til. For mange kan prosessen med å ringe inn en feil, være en mulighet til å lufte frustrasjoner rundt teknologien som har feilet, og vil oppleve det negativt at det ikke lenger er mulig. I tillegg vil ikke systemet lenger gi innmelder feedback på feilinnmeldingen. Systemet kan muligens videreutvikles slik at en KI-bot svarer innringer, men per nå blir det kun spilt av talemeldinger i begynnelsen av samtalen, og deretter er det kun input fra innringer. Som nevnt tidligere vil det bety at det ikke er noen mulighet til å styre innringer til å gi nødvendig informasjon om feilen, men det vil også bety at innringer ikke får feedback på at meldingen de har sendt inn blir hørt eller behandlet. Kvaliteten på feilmeldingen kan bli lavere hvis innmelder blir nedstemt av fraværet av menneskelig kontakt. Det kan forbedres noe med å for eksempel sende en SMS med beskjed om at melding er mottatt, men det vil ikke nødvendigvis være en løsning som løser hele problemet.

Vi mener at selv om produktet vi har utviklet har mangler, er det av samfunnsnyttig verdi fordi det kan bidra til økt produktivitet og lavere kostnader. Selv om produktet er laget for å være et alternativ til en tradisjonell servicedesk, er det ikke nødvendig at produktet overtar alle oppgaver og opererer alene. Det kan være at en hybridløsning vil gi best resultater, hvor det kan kuttes ned i behandlere, men fortsatt ha en liten kontingent for tilfeller hvor det er nødvendig med menneskelig innspill.

6 Konklusjon og videre arbeid

I løpet av utviklingsperioden har vi designet og utviklet et system for innsamling og analyse av feilmeldinger på vegne av Bodø kommune. Problemstillingen til prosjektet har vært om det er mulig å lage et automatisk system som samler inn og prosesserer feilmeldinger, og klassifiserer feilmeldingene etter bestemte kriterier. Vi mener at vi har lyktes med dette på flere områder, men at systemet fortsatt har mangler som gjør at det ikke kan brukes selvstendig til innsamling og klassifisering av feilmeldinger.

Det endelige produktet er delvis automatisert og leverer en transkribert feilmelding med analyse. Den eneste delen som trenger menneskelig input er valg av hendelse som skal analyseres, og der trengs det kun et tastetrykk for å få den ferdige analysen.

Produktet leverer en forbedring i forhold til dagens system på mange områder, men det er fortsatt enkelte mangler som gjør at det vil være vanskelig å sette det i produksjon og aktiv bruk. Det leverer raskt og billig ferdigklassifiserte feilmeldinger, sammen med løsningsforslag på feilen, basert på tidligere hendelser. Samtidig er det fortsatt et avvik mellom analysen som blir utført av RAG og klassifiseringen som blir gjort av Bodø kommune. Dette er til dels et resultat av at grunnlaget som brukes for å klassifisere feilen inneholder for lite informasjon om systemene, noe som gjør det vrient for RAG å identifisere korrekt system fra feilmeldingen, og til dels kan det skyldes at RAG ikke har vært optimalisert nok.

Transkriberingen fra tale til tekst er for det meste korrekt, men bommer på tjenestenavn og stedsnavn. Ettersom slik informasjon er viktig for å kunne bestemme klassifisering av feilen og hvor feilen har skjedd, kan det medføre vansker for korrekt klassifisering og feilretting.

6.1 Framtidig arbeid

Til framtidig arbeid, dersom vi skulle fortsette å utvikle produktet videre, ville vi fokusert mer på å forbedre analysen av feilmeldinger. Klassifiseringen av feil er veldig viktig for Bodø kommune, siden det avgjør hvordan feilen blir behandlet. Vi skulle derfor ønsket at klassifiseringene som ble gjort av produktet vårt, var nærmere Bodø kommunes vurderinger. Heldigvis ser vi at mye av argumentasjonen som blir gitt av KI er lik Bodø kommunes, og det virker ikke som det nødvendigvis er store endringer som må gjøres for å få klassifiseringen lik. En foreslått endring er å tydeliggjøre klassifiseringskriteriene, i tillegg til å inkludere flere beskrivelser. På den måten blir det flere holdepunkter for RAG ved søk i teksten, og det vil bidra ved tilfeller hvor transkribering ikke har gitt korrekt tjenestenavn men har inkludert beskrivelser av feilen.

6.1.1 Finjustering av NB-Whisper

Dersom et samarbeid med Bodø kommune lar seg etablere, kan finjustering av NB-Whisper-modellen utgjøre en hensiktsmessig strategi for å tilpasse talegjenkjenning til det fagspesifikke språket som benyttes innenfor kommunens administrative og operative rammeverk. Ved å trene modellen på domene-spesifikt materiale, vil den kunne lære å identifisere og korrekt transkribere sentrale

nøkkelord, lokale steds- og organisasjonsnavn, samt forkortelser og begreper som ofte er fraværende i generiske datasett.

Et praktisk fremgangsmåte for å muliggjøre slik finjustering innebærer innsamling av et representativt utvalg av innmeldinger, samtaler, transkriberinger eller lydopptak hvor det benyttes språk relatert til kommunens virksomhet. Disse datainnsamlingene kan deretter benyttes som treningsgrunnlag for modellen, med særlig vekt på å forbedre gjenkjenning av domene-spesifikke uttrykk. En slik prosess forventes å bidra til betydelig økt presisjon i transkripsjon av tale fra både ansatte og innbyggere, og kan dermed styrke kvaliteten på automatisert informasjonsbehandling og videre analysearbeid i kommunal sammenheng.

6.1.2 Overgang fra ACA Flask til FastAPI

Den nåværende ACA-løsningen benytter Flask som webapplikasjonsrammeverk. En sentral begrensning med Flask er mangelen på innebygd støtte for asynkron behandling, da Flask baserer seg på WSGI (Web Server Gateway Interface), som kun håndterer synkrone forespørsler. Den moderne asynkrone standarden er ASGI (Asynchronous Server Gateway Interface), som tillater samtidig håndtering av flere forespørsler uten å blokkere. Dette medfører at Flask krever at hver bruker må vente på at forrige bruker er ferdig før neste kan behandles, noe som kan føre til redusert ytelse og begrenset skalerbarhet ved høy belastning. En løsning på dette er å migrere fra Flask til FastAPI, et rammeverk som er designet for å støtte asynkronitet og høy ytelse, og som derfor er bedre egnet for produksjonsmiljøer med krav til skalerbarhet.

6.1.3 Eksperiment med ACA Jobs

Løsningene som benytter henholdsvis vanlig Azure Container Apps (ACA) og Azure Container Instances (ACI) fungerer godt, men vi har ikke i tilstrekkelig grad undersøkt et tredje alternativ: *Azure Container Apps Jobs*. Dette er en nyere funksjonalitet i ACA-plattformen som gjør det mulig å kjøre engangs- eller planlagte batch-jobber i stedet for permanente tjenester.

ACA Jobs kan være spesielt godt egnet for transkriberingsarbeid som utføres på forespørsel og avsluttes når jobben er fullført. I motsetning til vanlige ACA-applikasjoner, som kjører kontinuerlig og eksponerer HTTP-endepunkter, starter en ACA Job kun når den utløses manuelt, tidsdrevet eller basert på en hendelse. For vårt tilfelle vil manuelt eller hendelsesbasert utløsing være mest aktuelt.

Denne modellen kan gi lavere kostnader og bedre ressursutnyttelse ved at man kun betaler for kjøre- og oppstartstid. Videre gir det bedre kontroll over kjøretiden, siden man eksplisitt kan sette en maksimal tidsgrense for hver jobb.

Det ble gjort noen innledende tester med ACA Jobs, men på grunn av tidsbegrensninger og manglende støtte i det eksisterende automatiseringsoppsettet, ble det ikke valgt som hovedløsning. Likevel representerer ACA Jobs et interessant og potensielt mer effektivt alternativ for fremtidig implementasjon.

7 Liste over figurer

8 Liste over tabeller

9 Liste over akronymer og forkortelser

- ASR – Automatisk talegjenkjenning
- CAG – Cache Augmented Generation
- IaaS – Infrastructure as a Service
- IR – Information Retrieval
- KNN – K-nearest-neighbour
- KI – Kunstig intelligens
- LLM – Large language model
- LoRA – Low-Rank Adaptation
- NLP – Natural Language Processing
- NN – Nevrale nettverk
- NLP – Natural Language Processing
- PaaS – Platform as a Service
- PEFT – Parameter-effektiv finjustering
- RAG – Retrieval-augmented generation
- RNN – rekurrente nevrale nettverk
- SaaS – Software as a Service

Referanser

[1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. 3rd. Draft edition. 2023. URL: https://web.stanford.edu/~jurafsky/slp3/ (visited on 12/06/2023).

[2] Yunfan Gao et al. "Retrieval-Augmented Generation for Large Language Models: A Survey". In: (2024). arXiv: 2312.10997 [cs.CL]. URL: https://arxiv.org/abs/2312.10997.

[3] Alec Radford et al. "Robust Speech Recognition via Large-Scale Weak Supervision". In: (2022). arXiv: 2212.04356 [eess.AS]. URL: https://arxiv.org/abs/2212.04356.

[4] Axel Tidemann. *Nevrale Nettverk*. 2025. URL: https://snl.no/nevralt_nettverk.

[5] Ashish Vaswani et al. "Attention Is All You Need". In: (2023). arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[6] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: (2013). arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

[7] Ashish Vaswani et al. "Attention is All You Need". In: *Advances in neural information processing systems*. Vol. 30. 2017.

[8] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[9] Geoffrey Hinton et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[10] Alex Graves et al. "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks". In: vol. 2006. Jan. 2006, pp. 369–376. DOI: 10.1145/1143844.1143891.

[11] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *arXiv preprint arXiv:1801.06146* (2018).

[12] Edward Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *arXiv preprint arXiv:2106.09685* (2021).

[13] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*. Vol. 463. ACM press New York, 1999.

[14] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[15] Brian J Chan et al. "Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks". In: *arXiv preprint arXiv:2412.15605* (2025). License: CC BY-SA 4.0. arXiv: 2412.15605 [cs.CL]. URL: https://arxiv.org/abs/2412.15605.

[16] Shunyu Yao et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023. arXiv: 2210.03629 [cs.CL]. URL: https://arxiv.org/abs/2210.03629.

[17] Hugging Face. *How to Generate Text with Transformers*. Accessed: 2025-05-20. 2023. URL: https://huggingface.co/blog/how-to-generate.

[18] Alex Graves. *Sequence Transduction with Recurrent Neural Networks*. 2012. arXiv: 1211.3711 [cs.NE]. URL: https://arxiv.org/abs/1211.3711.

[19] T. M. Cover and P. E. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27.

[20] P. Lewis et al. “Retrieval-augmented generation for knowledge-intensive NLP tasks”. In: *arXiv preprint arXiv:2005.11401* (2020). URL: https://arxiv.org/abs/2005.11401.

[21] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 1999.

[22] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL]. URL: https://arxiv.org/abs/1310.4546.

[23] J. MacQueen. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1. University of California Press. 1967, pp. 281–297.

[24] Michael Armbrust et al. “A view of cloud computing”. In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[25] Dirk Merkel. “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux journal* 2014.239 (2014), p. 2.

[26] Brendan Burns et al. “Borg, Omega, and Kubernetes”. In: *Communications of the ACM*. Vol. 59. 5. 2016, pp. 50–57.

[27] Ken Schwaber and Jeff Sutherland. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Available at: https://scrumguides.org/scrum-guide.html. 2020.

[28] Rozilawati Razali and Mashal Alqudah. “An Empirical Study of Scrumban Formation based on the Selection of Scrum and Kanban Practices”. In: *International Journal on Advanced Science, Engineering and Information Technology* (2018).

[29] Corey L. Lewis. “Scrumban: Essays on Kanban Systems for Lean Software Development”. In: *Modus Cooperandi Press* (2010). Available at: https://www.scrumplop.org/wiki/Scrumban.

[30] David J. Anderson. “Kanban: Successful Evolutionary Change for Your Technology Business”. In: *Blue Hole Press* (2010).

[31] Microsoft Corporation. *What is Azure Communication Services?* Accessed: May 11, 2025. Microsoft Learn. 2024. URL: https://learn.microsoft.com/en-us/azure/communication-services/overview.

[32] Per E Kummervold et al. “Whispering in Norwegian: Navigating Orthographic and Dialectic Challenges”. In: *arXiv preprint arXiv:2402.01917* (2024). arXiv: 2402.01917 [cs.CL]. URL: https://arxiv.org/abs/2402.01917.

[33] NB-AI Lab. *NB-Whisper Large: Norwegian Automatic Speech Recognition Model*. https://huggingface.co/NbAiLab/nb-whisper-large. Accessed: 2025-05-19. 2023.

[34] *Service Level Agreement for Microsoft Online Services March 1, 2025*. URL: https://www.microsoft.com/licensing/docs/view/Service-Level-Agreements-SLA-for-Online-Services?lang=22.

[35] Alec Radford, Jong Wook Gao, et al. *Whisper - transcribe.py lines 86–89*. https://github.com/openai/whisper/blob/main/whisper/transcribe.py\#L86-L89. Accessed: 2025-05-21. 2022.

[36] Sahara Muhaisen. *Regjeringen ber 80 prosent av offentlig sektor bruke KI innen 2025*. Accessed 16.05.2025. URL: https://www.nrk.no/norge/regjeringen-vil-at-80-prosent-av-offentlig-sektor-bruker-ki_-_-urealistisk_-mener-ki-forsker-1.16843972.

[37] Perry Chen and Aziz Hasan. *The Future of Crowdfunding Creative Projects*. Accessed 16.05.2025. URL: https://www.kickstarter.com/articles/the-future-of-crowdfunding-creative-projects?ref=section-articles-lets-build-whats-next-for-crowdfunding-creative-projects-article-body.

[38] Leo Schwartz. *The untold story of Kickstarter's crypto Hail Mary—and the secret $100 million a16z-led investment to save its fading brand*. Accessed 16.05.2025. URL: https://finance.yahoo.com/news/untold-story-kickstarter-crypto-hail-120000205.html.

Vedlegg

1. Oppgavetekst
2. Forprosjektrapport
3. Kravdokument
4. Systemdokumentasjon
5. Prosessdokument