Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Ingvild Andersen, Jon Akselberg Langholm og Erik Flæsen Dalen

MLOps på Google Cloud Platform

Bacheloroppgave i Informatikk, drift av datasystemer
Veileder: Jostein Lund
Medveileder: Torgeir Thoresen
Mai 2021

NTNU
Kunnskap for en bedre verden

Ingvild Andersen, Jon Akselberg Langholm og Erik Flæsen Dalen

MLOps på Google Cloud Platform

Bacheloroppgave i Informatikk, drift av datasystemer
Veileder: Jostein Lund
Medveileder: Torgeir Thoresen
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Innhold

Sammendrag 2
Abstract 2

1 Forstudierapport **4**

2 Designrapport **43**

3 Driftsrapport **80**

4 Sluttrapport **128**

Sammendrag

Statens vegvesen samler inn store mengder data fra norske veier. En satsning for å ta i bruk mer moderne teknologi har naturligvis medført et ønske om å ta i bruk maskinlæringsmodeller i sine prosjekter. Utvikling av maskinlæringsmodeller er en prosess der modellene går gjennom flere iterasjoner med trening på data. Uten gode kontrollmekanismer på dataen og versjonering, kan det oppstå komplikasjoner. Derfor er det et viktig hjelpemiddel å innføre MLOps; standardiserte prosesser for maskinlæring ende til ende. Prinsippene i MLOps kan implementeres ved hjelp av produksjonspipeliner. Hensikten med denne oppgaven er å foreslå konkret hvordan en produksjonspipeline kan innføres og tas i bruk hos Statens vegvesen. Bachelorgruppen har utviklet en slik pipeline som innfører MLOps på Google Cloud Platform. Denne rapporten viser hvordan bachelorgruppen kom frem til, designet rundt og innførte bruken av teknologier som Kubeflow Pipelines, TensorFlow Extended og Google Cloud AI Platform. Resultatet er en generisk pipeline som kan tilpasses en hvilken som helst TensorFlow-modell, som skalerer godt, og som integreres med Google Cloud-tjenester i hvert ledd.

Abstract

The Norwegian Public Roads Administration is collecting vast amounts of data from Norwegian public roads. An interest in utilising more modern technology has brought with it a desire to apply machine learning models to their projects. The development of machine learning models is a process that brings the models through several iterations of testing and training on big amounts of data. Complications quickly arise without good mechanisms for tracking data and versions. Therefore, implementing the principles of MLOps is of great help to anyone dealing with this. The purpose of this paper is to deliver a concrete proposal as to how machine learning production pipelines can be applied by the Norwegian Public Roads Administration in their projects. The bachelor's thesis group has developed such a pipeline that applies the principles of MLOps on Google Cloud Platform. This paper will showcase how the authors researches, designed around, and applied technologies such as Kubeflow Pipelines, TensorFlow Extended and Google Cloud AI Platform. The result is a generic pipeline that can be customised to fit any TensorFlow machine learning model, that scales well, and that integrates Google Cloud functions in every step.

1 Forstudierapport

MLOps på Google Cloud Platform
Forstudierapport

Ingvild Andersen, Jon Akselberg Langholm, Erik Flæsen Dalen

13. mai 2021

Statens vegvesen

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
02.02.2021	1.0	Første versjon av forstudierapport	Erik F.D., Ingvild A. og Jon L.
05.02.2021	1.1	Mindre revisjon med bakgrunn i tilbakemelding fra veileder Torgeir.	Jon L.
13.05.2021	1.2	Språkvask, mindre endringer i setningsoppbygginger	Erik F. D.

Innhold

1 Introduksjon **5**
1.1 Oversikt over dokumentet 5

2 Bakgrunn **6**
2.1 Dagens system og rutiner 7

3 Prosjektmål **9**
3.1 Effektmål 9
3.2 Resultatmål 9
3.3 Prosessmål 10
3.4 Prosjektets omfang 10
3.4.1 Prosjektgruppen skal 11
3.4.2 Prosjektgruppen skal ikke 11
3.4.3 Videre begrensning av omfang 12
3.5 Produktets funksjonelle egenskaper 12
3.5.1 Ikke-funksjonelle egenskaper og krav 16
3.6 Milepæler og hovedaktiviteter 17

4 Interessenter og rammebetingelser **19**
4.1 Interessentanalyse 19
4.2 Rammebetingelser 20

5 Kritiske suksessfaktorer **22**
5.1 Suksessfaktorer 22
5.2 Informasjonsbehov 23

6 Risikoanalyse **25**

7 Kost/nytte-analyse **28**
7.1 Kvantifiserbar og ikke-kvantifiserbar nytte 29
7.1.1 Kvantifiserbar nytte 29
7.1.2 Ikke-kvantifiserbar nytte 29
7.2 Bortfall av direkte kostnader 30
7.3 Estimerte kostnader 31
7.4 Sammenstilling av kost/nytte 31

8 Retningslinjer og standarder **32**
8.1 Krav til dokumentasjon 32
8.2 Krav til kvalitetsgjennomganger 32
8.3 Krav til standarder og metoder 33
8.4 Endringshåndtering 33

9 Prosjektorganisering **35**

10 Anbefaling om videre arbeid **37**

Referanser **38**

Figurer

2 Dagens system 8
3 Milepæler for prosjektet 18
4 Stipulert arbeidsprosess 18
5 Interessenter 20
6 Suksessfaktorer 22
7 Informasjonsbehov 24
8 Risikoer 25
9 Risikoer i risikotabell 26
10 Strategier for å håndtere risikomomenter 27
11 Prosjektorganisering 35

1 Introduksjon

Alle prosjekter begynner med en idé, men ikke alle idéer bør bli prosjekter. Forstudiens hensikt er å kartlegge om en prosjektidé lar seg realisere innen gitte ressurs-, tids- og kostnadsrammer. Formålet er å avdekke om prosjektet kan gjennomføres slik oppdragsgiver ønsker, eventuelt hvilke endringer som må til for å støtte opp om gjennomførbarhet. Dokumentet skaper en avtale mellom oppdragsgiver og oppdragstaker, og bidrar til at begge parter sitter på samme type informasjon. Forstudien beskriver også hvordan det planlegges å gjennomføre prosjektet, og skal fremstilles slik at den finnes like forståelig for leser som for forfatter. I denne konteksten sitter oppdragsgiver på mye faglig kunnskap, så faglige begreper kan benyttes. Forstudierapporten gir en overordnet oversikt over prosjektets forventede resultat, men det tas forbehold om at resultatet kan endre seg gjennom den videre prosjektperioden.

1.1 Oversikt over dokumentet

Forstudien introduseres med **dokumentets hensikt (1)** og en kort **oversikt (1.1)**. Videre presenteres **bakgrunnen for prosjektet (2)**, nærmere bestemt en overordnet beskrivelse av bedriften, deres situasjon, problemer og ønsker. Neste del av dokumentet går inn på **prosjektmål (3)**, helt konkret **effektmål (3.1)**, **resultatmål (3.2)** og **prosessmål (3.3)**. Avsnittet beskriver også **prosjektets omfang (3.4)**, som avgrenser hva prosjektgruppen skal og ikke skal gjøre gjennom prosjektperioden. Her synliggjøres også **produktets funksjonelle egenskaper (3.5)** og **milepæler og hovedaktiviteter (3.6)**. I fjerde avsnitt analyseres **interessenter og rammebetingelser (4)**. Her informeres leser om hvem som er de involverte i prosjektets gjennomførelse og resultat, samt prosjektets rammebetingelser. Midtveis i forstudien foreligger prosjektets **kritiske suksessfaktorer (5)**, i tillegg til en oversikt over prosjektets **informasjonsbehov (5.2**). Dette er faktorer som kreves for at prosjektet skal lykkes, og dokumentasjon som støtter opp under disse. Femte avsnitt omfatter en **risikoanalyse (6)**, som setter søkelys på hendelser som kan ha negativ innvirkning på prosjektresultatet, faren for at disse skal finne sted, samt tiltak for å redusere konsekvensen av hendelsene. Neste

avsnitt går inn på forstudiens **kost/nytte-analyse (7)**, der man forsøker å tallfeste både nytten og kostnaden av en prosjektgjennomføring. Da disse verdiene er minimale og vanskelige å tallfeste i denne forstudien, er sammenstillingen av kost/nytte uteblitt. Videre går rapporten inn på **retningslinjer og standarder (8)**. Her presenteres krav til dokumentasjon, kvalitertsgjennomganger, standarder, metoder og endringshåndtering. Til slutt får lesere en oversikt over **prosjektorganisasjonen (9)**, som beskriver hvem som er med i prosjektet og deres arbeidsfordeling, samt en **anbefaling om videre arbeid (10)**.

2 Bakgrunn

Statens vegvesen er et nasjonalt forvaltningsorgan underlagt Samferdselsdepartementet. De har ansvar for å forvalte, planlegge, bygge, drifte og vedlikeholde de norske riksveiene.[1] Organisasjonen ønsker å bli mer datadrevet, som har ført til utvikling av dataplattformen Saga. Teamet bak plattformen består av interne ansatte fra Statens vegvesen, samt eksterne konsulenter fra selskapene Bekk og Webstep. Den underliggende infrastrukturen kjører på Google Cloud Platform. Deres oppgave er å utvikle forskjellige moduler i denne dataplattformen for å kunne samle inn og analysere store datamengder. I dag utføres eksempelvis mange manuelle oppgaver på oppdrag fra Statens vegvesen som i teorien kan automatiseres ved hjelp av maskinlæringsmodeller som trenes på disse datamengdene. En automatisering av manuelle arbeidsoppgaver kan frigi ressurser og tidsbruk, som Vegvesenet kan prioritere på andre områder. Målet er at de skal kunne bruke maskinlæringsmodeller for å gjøre forutsigelser rundt diverse formål som omfatter forvaltningsorganets virke. Oppdragsgiver ønsker å få forslag på hvordan man effektivt kan sette disse modellene i produksjon, og om dette er teknisk mulig. Dette beskrives videre som en produksjonspipeline.

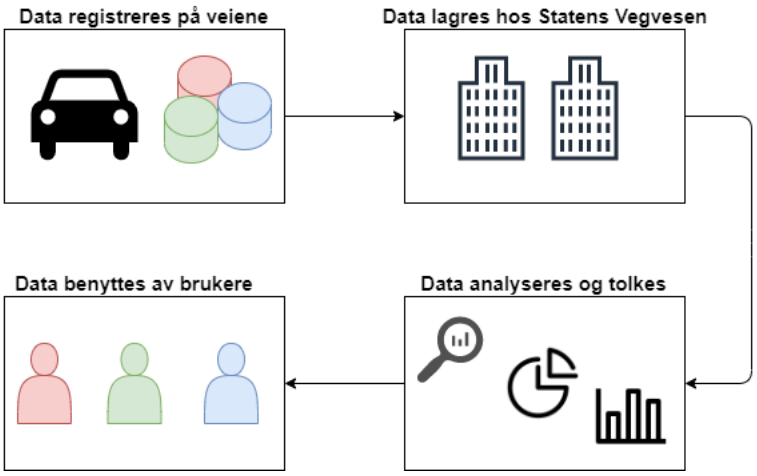
Med bakgrunn i dette behovet, har prosjektgruppen fått som oppgave å utarbeide en 'Proof of Concept' på en produksjonspipeline for en maskinlæringsmodell i Google Cloud Platform. Denne plattformen ønsker oppdragsgiver at skal brukes i flest mulig steg av produksjonspipelinen. Bedriften ønsker forslag som kan vise hvordan konseptet kan gjennomføres

konkret. Dette innebærer klart definerte suksesskritierer, dokumentasjon for gjennomførelse, en evalueringskomponent og et forslag for videre arbeid. Et 'Proof of Concept' innebærer også at man identifiserer potensielle tekniske og logistiske problemer som kan komme i veien for produksjonsprosessen.

2.1 Dagens system og rutiner

Dagens løsning eksisterer ikke, og innehar derfor ingen identifiserbare problemer. Per i dag samler Statens vegvesen inn store mengder data som kan hentes ut, leses og analyseres manuelt eller av programvare. Idéen om en løsning som går ut på å bruke disse datamengdene til å trene opp en maskinlæringsmodell for automatisering av arbeidsoppgaver, er nylig påbegynt. Behovet internt er å skape en produksjonspipeline for å standardisere prosessen ved bruk av maskinlæring fra første stund.

Selv om dagens system ikke eksisterer, er det i planleggingsfasen, orkestrert av teamet bak dataplattformen Saga. Basert på de store mengdene innhentet data skal maskinlæringsmodeller utvikles og trenes av enten innleide datavitere eller interne ansatte i Statens vegvesen. Disse maskinlæringsmodellene skal trenes og videreutvikles i Google Cloud Platform. Når modellen blir satt i produksjon, skal den kunne innhente nye datamengder. Disse datamengdene må valideres og kontrolleres, samt førprosesseres slik at de kan leses av pipelinen. Modellen må kunne trenes, justeres og evalueres slik at den alltid fungerer optimalt. Saga eies som nevnt av Statens vegvesen, som også finansierer videreutviklingen av prosjektet. Når dette prosjektet er ferdigstilt, vil produktet overtas av Saga, og videreutvikles utover dette prosjektets omfang.



Figur 2: Dagens system

Data samles inn av sensorer eller manuelle oppgaver, for så at disse lagres i store datalagre hos Statens vegvesen. Datamengdene analyseres og tolkes, før de presenteres for brukerne.

3 Prosjektmål

Prosjektmålene skal være en felles forståelse mellom oppdragsgiver og oppdragstaker. De skal lede begge parter og prosjektet mot felles mål. Målene er utarbeidet i samarbeid med oppdragsgiver, slik at man opprettholder en felles forståelse gjennom hele prosjektperioden. Prosjektmålene er delt inn i effektmål, resultatmål og prosessmål.

3.1 Effektmål

Effektmålene beskriver den endringen Statens vegvesen ønsker å få ut av prosjektet. Dette er prosjektets langsiktige virkninger for virksomheten.

- Ta i bruk mer moderne teknologi i veisektoren.
- Innhente forslag til hvordan de kan kjøre det foreslåtte systemet i produksjon.
- Et konkret produkt som kan testes ut.
- Kunne vurdere om dette er retningen de ønsker å bevege seg i, eller om de vil utforske alternativer.

3.2 Resultatmål

Resultatmål beskriver hvilke resultater som skal foreligge når prosjektet er ferdigstilt, samt hva som må gjøres for å oppnå disse resultatene. De gir altså en pekepinn på hva som konkret må gjøres.

Hovedmål:

- Et ferdigstilt 'Proof of Concept' på en produksjonspipeline for en maskinlæringsmodell i GCP.
- Ha resultatet klart innen 20.05.2021

Delmål:

- Kunne kjøre ut en maskinlæringsmodell i GCP.
- Kunne kjøre ut nye versjoner av en maskinlæringsmodell i GCP.

- Kunne eksponere en maskinlæringsmodell for input og bruke den fra andre applikasjoner.
- Finne ut hvordan man kan overvåke en maskinlæringsmodell og dens evne til å predikere riktig.
- Finne ut hvordan man kan skalere en pipeline med hensyn på kapasitet og ytelse.

Merknad: Det er ikke et mål i seg selv å ferdigstille alle punktene i oppgavebeskrivelsen eller alle disse delmålene, men å gjennomføre det prosjektgruppen evner på best mulig måte.

3.3 Prosessmål

Prosessmålene handler om hvilken verdi som skal oppnås som følge av selve prosjektprosessen. Disse målene består av både individuelle og kollektive forventninger til prosjektet, i tillegg til hva hvert enkelt gruppemedlem forventer å sitte igjen med etter endt prosjektperiode. Disse målene er kompetansebasert og består av studie-, sosiale- og tekniske kompetansemål.

- Økt kunnskap innen GCP og dens underliggende verktøy.
- Kompetanse på hvordan man planlegger og innfører en produksjonspipeline for maskinlæringsmodeller i GCP.
- Styrkede samarbeidsevner i teamarbeid med færre gruppemedlemmer.
- Økt kunnskap om individuelle styrker og svakheter i gruppesammenheng
- Erfaring rundt kommunikasjon og samarbeid med en ekstern, reell bedrift
- Høy faglig måloppnåelse

3.4 Prosjektets omfang

Denne delen av forstudien avklarer omfanget av prosjektet, og det er her man avgrenser oppgaven. En slik avgrensning fører til at oppdragsgiver og oppdragstaker er samstemte om nøyaktig hva prosjektet omfatter. Dette bidrar til at prosjektgruppen ikke bruker unødige ressurser på spesifikasjoner

som ikke inngår i prosjektomfanget. Produksjonspipelinen skal effektivisere ressursbruken i Statens vegvesen, og den legger til rette for at organisasjonen får utnyttet sine datamengder. Dette krever klare suksesskriterier for hvordan pipelinen skal operere, samt nøye utarbeidet dokumentasjon slik at oppdragsgiver enkelt og effektivt kan fortsette videre arbeid. Det er viktig å spesifisere at prosjektgruppen ikke har som ansvar å implementere eller drifte produksjonspipelinen som utarbeides, ei heller å utvikle selve maskinlæringsmodellen. Det skal heller ikke tillegges funksjonalitet som ikke er relevant for oppgavebeskrivelsen.

3.4.1 Prosjektgruppen skal

- Utarbeide et 'Proof of Concept' på en produksjonspipeline for en maskinlæringsmodell i Google Cloud Platform tilpasset behov hos Statens vegvesen
- Definere klare suksesskriterier for produksjonspipelinen
- Dokumentere hvordan 'Proof of Concept' gjennomføres slik at bedriften kan gjenskape resultatene
- Komme med forslag om videre arbeid om 'Proof of Concept' viser seg vellykket

3.4.2 Prosjektgruppen skal ikke

- Utvikle en maskinlæringsmodell
- Drifte produksjonspipeline
- Implementere produksjonspipeline
- Legge til funksjonalitet som ikke er relevant for oppgavebeskrivelsen
- Drive opplæring i nye systemer
- Sette opp miljø i Google Cloud Platform

3.4.3 Videre begrensning av omfang

Det er viktig å påpeke at oppgaven er bred, og vil sannsynligvis avgrenses i enda større grad i etterkant av forstudien. Dette er avklart og vil skje i samarbeid med oppdragsgiver. Bakgrunnen for dette er at omfanget av oppgaven også er ukjent for oppdragsgiver, da de heller ikke har tidligere erfaring med å gjennomføre et slikt prosjekt. Derav behøves det et 'Proof of Concept'. Om det viser seg at man må skalere ned omfanget, vil dette foregå etter den prioriterte oppgavebeskrivelsen som ble gitt ved prosjektstart. I prioritert rekkefølge:

1. Deploye maskinlæringsmodeller i GCP
2. Deploye nye versjoner av samme maskinlæringsmodell
3. Eksponere maskinlæringsmodeller for input
4. Bruke maskinlæringsmodeller fra andre applikasjoner
5. Overvåke maskinlæringsmodellen og dens evne til å predikere riktig
6. Skalere pipeline med hensyn på kapasitet/ytelse
7. Finne ut hvordan man kan trene maskinlæringsmodeller effektivt
8. Finne ut hvordan man kan lagre maskinlæringsmodeller på en hensiktsmessig måte

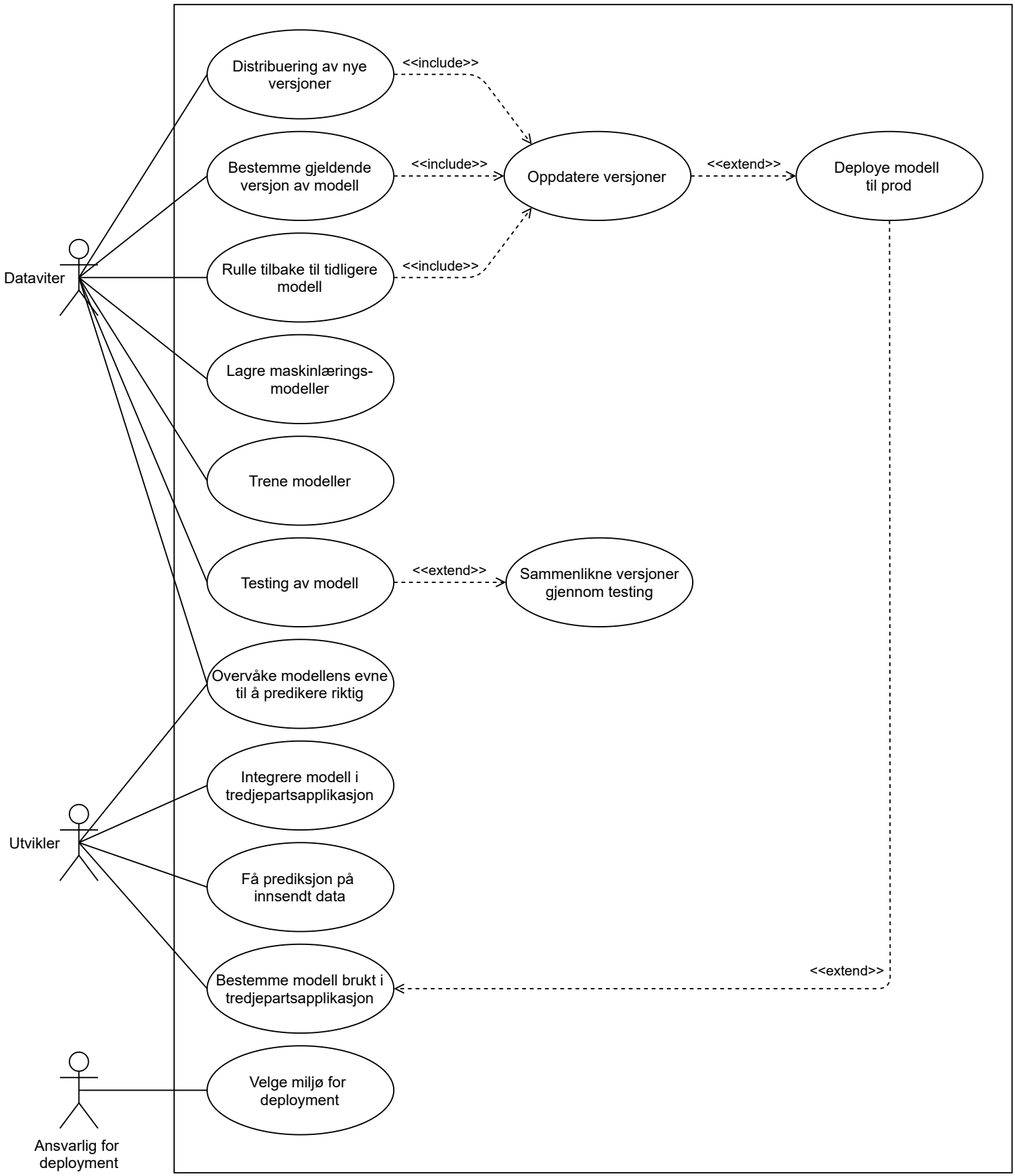
Om prosjektet må begrenses, sløyfes funksjonalitet nedenfra og opp.

3.5 Produktets funksjonelle egenskaper

En rekke verktøy må anvendes for å gjennomføre prosjektet. Det skal i hovedsak utarbeides i GCP, og prosjektgruppen vil da unytte denne plattformens funksjonalitet. De funksjonelle egenskapene til produktet kommer av punktene i kapittelet *Videre begrensning av omfang*. Produktet må altså kunne motta maskinlæringsmodeller, inndata og utdata. For å få til denne funksjonaliteten, skal forskjellige komponenter i GCP settes sammen. Dette beskrives grundigere i designrapporten, men i all hovedsak skal GCP kunne dekke all funksjonaliteten som behøves for å kunne utvikle produktet. GCP har blant annet, gjennom verktøyet *AI Platform*, innebygget funksjonalitet

for å deploye maskinlæringsmodeller, trene dem og gjøre prediksjoner. Disse oppgavene kan automatiseres ved hjelp av verktøyene *Cloud Functions* og *Pub/Sub*. Andre aktuelle verktøy i GCP er *BigQuery* og *Kubeflow*.

3.5.0.1 Use-case-diagram


Distribuering av nye versjoner
<<include>>
Bestemme gjeldende versjon av modell
<<include>>
Oppdatere versjoner
<<extend>>
Deploye modell til prod
Dataviter
Rulle tilbake til tidligere modell
<<include>>
Lagre maskinlærings-modeller
Trene modeller
Testing av modell
<<extend>>
Sammenlikne versjoner gjennom testing
Overvåke modellens evne til å predikere riktig
Integrere modell i tredjepartsapplikasjon
Utvikler
Få prediksjon på innsendt data
Bestemme modell brukt i tredjepartsapplikasjon
<<extend>>
Velge miljø for deployment
Ansvarlig for deployment

Samlet: 18

3.5.0.2 Forklaring av use-caser

- **Distribuering av nye versjoner**
  Dataviter skal ha metoder for å distribuere og oppdatere versjoner av modeller.
- **Bestemme gjeldende versjon av modell**
  Dataviter skal kunne bestemme hvilken modell som er gjeldende i det gitte prosjektet. Dette er den modellen som skal trenes og testes.
- **Rulle tilbake til tidligere modell**
  Dataviter skal kunne rulle tilbake til en tidligere versjon av modellen. Dette er i tilfeller der det skulle være feil med den nye modellen.
- **Oppdatere versjoner**
  De tre use-casene over er alle avhengig av et versjonsstyringssystem. Dette systemet må ha kontroll på hvilke versjoner som eksisterer av hvilke modeller, hvordan disse er trent slik at man kan reprodusere treningen, holde styr på hvilke modeller som trenes, og hvilke som er i produksjon.
- **Produksjonssette ny modell**
  Dataviteren skal deploye trente modeller til produksjon. Når en modell er deployet til produksjon, er den tilgjengelig for andre gjennom API-kall.
- **Lagre maskinlæringsmodeller**
  Dataviteren skal lagre nye maskinlæringsmodeller. Modeller skal være adskilt fra hverandre slik at ikke modeller som gjør forskjellige oppgaver blandes sammen. Modellene har adskilte datasett som heller ikke skal blandes på tvers av modeller.
- **Trene modeller**
  Dataviter skal kunne trene modellene som ligger i pipelinen. Trente modeller lagres som nye versjoner og med en forklaring på hva de er trent på og hvilken versjon de er trent fra.
- **Testing av modell**
  Modellene skal kunne testes på testdata for å undersøke hvordan de presterer.

- **Sammenlikne versjoner gjennom testing**
  Dataviteren skal kunne kjøre to modeller på det samme testdatasettet for å sammenligne de to og se hvor godt de gjør det i forhold til hverandre på de samme dataene.
- **Overvåke modellens evne til å predikere riktig**
  Dataviter og utvikler skal kunne se modellenes nøyaktighet. For utvikler er dette spesielt relevant når vedkommende skal velge hvilken modell som skal brukes i applikasjonen.
- **Integrere modell i tredjepartsapplikasjon**
  Utvikler skal kunne bruke prediksjoner fra en maskinlæringsmodell i sin applikasjon. Dette kan skje gjennom et API-kall i programkoden til utvikleren.
- **Få prediksjon på innsendt data**
  Utvikler skal kunne sende inn mengder data til modellen, og få prediksjoner på denne.
- **Bestemme modell brukt i tredjepartsapplikasjon**
  Utvikler skal ha muligheten til å selv bestemme hvilken modell som skal brukes i vedkommendes applikasjon. De tilgjengelige modellene er da de som datautvikleren har deployet til produksjon. Det skal være valgfritt for utvikleren å velge modell selv. Det skal være mulighet for å automatisk velge den nyeste modellen som dataviteren har deployet.
- **Velge miljø for deployment**
  Ansvarlig for deployment skal bestemme hvilket miljø/prosjekt modellen skal deployes til.

3.5.1 Ikke-funksjonelle egenskaper og krav

Det er en rekke krav til funksjonalitet som ikke direkte passer til en use-case.

- **Deploying av modeller**
  Når en ny modell skal deployes, skal man være trygg på det man deployer. Dette innebærer å være trygg på at man deployer noe som fungerer. Man skal sikre seg at man deployer noe som har den funksjonaliteten som forventes slik at det ikke skapes problemer for

brukeren av modellen, slik som utvikleren. Man må også være trygg på at modellen har god nok ytelse til det den skal brukes til.

- **Integrering med tredjepartsapplikasjoner**
  Utvikleren av tredjepartsapplikasjoner skal slippe å forholde seg til maskinlæringsmodellen utover å implementere API-kallet. Nye modeller må ikke endre på formatet på svaret til API-kallet. Utvikleren må ikke endre koden sin til stadighet for å tilpasse seg modellene.
- **Versjonskontroll**
  Versjoner av modeller bør lagres på en hensiktsmessig måte, og versjoneringen av modeller bør være logisk.
- **Mating av data**
  Treningsdata og testdata til modellene må lagres på en hensiktsmessig måte. Treningsdataen kan gjerne skilles på eksempelvis størrelse, og testdataen kan for eksempel skilles på spesielle forhold/scenarier.

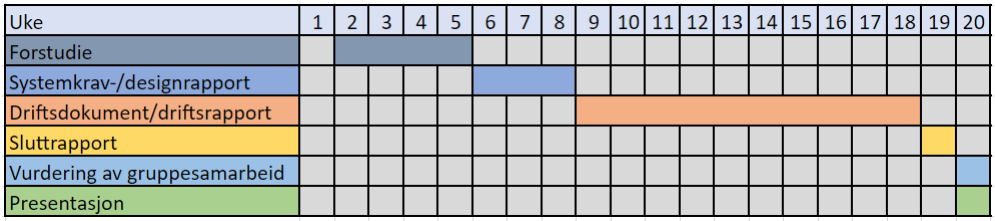
3.6 Milepæler og hovedaktiviteter

Følgende del av dokumentet beskriver hvordan prosjektgruppen forventer å disponere tiden gjennom prosjektperioden. Dette omfatter en oversikt over milepæler, som gir en overordnet oversikt over hva som må dokumenteres, og til hvilken tid dette forventes gjennomført. Milepælene har blitt satt med utgangspunkt i hovedaktivitetene, som består av forstudierapport, designrapport, driftsrapport og sluttrapport. Det utarbeides i tillegg en prosjekthåndbok fortløpende. Se tabell for detaljert oversikt over milepæler og tidsbruk, samt figur 4 for en grafisk fremstilling av stipulert arbeidsprosess.

Planlagte milepæler for prosjektet er:

Milepæler	Start	Slutt
Forstudierapport	11.01.2021	01.02.2021
Designrapport	01.02.2021	28.02.2021
Driftsrapport	01.03.2021	09.05.2021
Sluttrapport	10.05.2021	16.05.2021
Gruppevurdering	17.05.2021	20.05.2021
Innlevering	20.05.2021	20.05.2021

Figur 3: Milepæler for prosjektet



Figur 4: Stipulert arbeidsprosess

For en mer detaljert oversikt, se Gantt-diagram i prosjekthåndboken.[2]

Tilgjengelighet: Prosjektgruppen tar påskeferie deler av uke 13, og vil da være mindre tilgjengelige enn vanlig. Det samme gjelder dagene som leder opp til eksamen 11. mai. Dette gjenspeiles ikke i Gantt-diagram eller stipulert arbeidsprosess.

4 Interessenter og rammebetingelser

I ethvert prosjekt er det nødvendig å kartlegge interessenter for å tydeliggjøre de enkelte interessentene, hvordan de påvirker prosjektet og hvordan de selv blir påvirket av prosjektet og/eller dets resultater. Samtidig stadfestes interessentenes forventninger og i hvilken grad interessentene selv kan forventes å bidra i prosjektet. Rammebetingelser beskriver absolutte krav som stilles til prosjektets gjennomføring, som vil ha avgjørende innflytelse på planer og valg i prosjektet. Dermed er det, ved å supplere en interessentanalyse med konkrete rammebetingelser, enklere å planlegge videre gjennomføring av prosjektet.

4.1 Interessentanalyse

Interessentene i dette prosjektet er personer, grupper eller organisasjoner som kan påvirke eller vil bli påvirket av prosjektets gjennomføring og/eller resultat. Statens vegvesen v/ produkteier Lars Meisingseth er oppgavestiller, og vil gjennom prosjektet ha kontinuerlig kommunikasjon med prosjektgruppen. Dette prosjektet ønsker Statens vegvesen å gjennomføre i fremtiden, så et Proof of Concept viser mulige løsninger. De stiller med en intern veileder (Torgeir Thoresen), som vil bidra med teknisk veiledning og fungere som sparringspartner. Samtidig stiller de med en intern sensor (Safurudin Mahic), som vil bidra med sensur av bacheloroppgaven. Ut over dette bidrar Statens vegvesen med nødvendige lisenser og programvare. Faglig veileder, Jostein Lund v/NTNU, er prosjektgruppens faglige kontakt. Veileder skal svare på spørsmål tilknyttet rapportering og prosjektgjennomføring, samt fungere som sensor ved vurdering av bacheloroppgaven. Prosjektgruppen, bestående av Jon Langholm, Ingvild Andersen og Erik Flæsen Dalen, er ansvarlige for prosjektet. Herunder faller utvikling, rapportskriving, kommunikasjon og dokumentering. Utviklere er her definert til å være brukere av selve prediksjonsdataen Statens vegvesen presenterer. Dette vil typisk være organisasjoner som ønsker å benytte tjenesten(e) Statens vegvesen leverer, eller Statens vegvesen selv, til å utvikle tredjepartsapplikasjoner eller andre applikasjoner som nytter prediksjonsdata.

Under følger en analyse av aktuelle interessenter, med fokus på hva interessenten ønsker av prosjektet, hvilken innflytelse interessenten har på prosjektet med tanke på prosess og/eller produkt, hva interessenten forventes å bidra med og hvordan hindringer kan reduseres.

Interessent	Interesse i prosjektet	Innflytelse (H,M,L)	Bidrag til prosjektet	Strategi for å redusere hindringer
Statens vegvesen	Fungerende pipeline som kan brukes i SVV, teknologisk utvikling	H (Høy)	Teknisk kompetanse, løsnings-orientering, veiledning, sensur, beslutninger	Lav-terskel kommunikasjon på Slack, Stand-up-møter ca to ganger i uken, møter hver 2.-3. uke
Faglig veileder (NTNU)	Høy måloppnåelse for studentene	H (Høy)	Prosess-veiledning, vurdering, veiledning	Møter hver 2.-3. uke
Prosjekt-gruppen	Lære om tema, høy måloppnåelse, vellykket produkt	H (Høy)	Ansvar	Daglig kommunikasjon og samarbeid, gruppekontrakt, god planlegging
Utviklere	Tilgang til SVVs ML-modeller	M (Middels)	Presentere funksjonelle ønsker	Løpende kommunikasjon

Figur 5: Interessenter

4.2 Rammebetingelser

Rammebetingelser er definert som absolutte krav til prosjektet, som har direkte og avgjørende innflytelse på planer og valg i prosessen. Betingelsene er som følger:

- Google Cloud Platform benyttes til lagring, trening og analyse av maskinlæringsmodeller
- GCP-miljø settes opp av Statens vegvesen

- Kostnader tilknyttet Google Cloud Platform har en myk grense på $500 per måned per prosjekt
- Studentene skal jobbe 500 timer +/- 5% hver, som gir mellom 1425 og 1575 timer totalt
- Prosjektstart 5. januar 2021
- Prosjektet skal være ferdigstilt og all dokumentasjon levert innen 20. mai 2021
- Prosjektgruppens spisskompetanse er relativt lav i utgangspunktet

5 Kritiske suksessfaktorer

De kritiske suksessfaktorene er faktorer der et gunstig utfall er avgjørende for at prosjektet skal kunne anses som vellykket for alle interessenter. De rettleder arbeidsprosessen, og setter fokus på kritisk viktige forhold hos både oppdragsgiver og oppdragstaker.

5.1 Suksessfaktorer

Suksessfaktor	Kommentar
Aktiv involvering og tilrettelegging fra SVV	Nødvendig for å sikre at SVV får et produkt som stemmer overens med forventningene deres, samt tilrettelegging med tanke på nødvendige tilganger/programvare/kunnskap
Tilgjengelige veiledere	Nødvendig for prosjektgruppen å få tilbakemelding underveis
Tilgjengelighet på GCP	Nødvendig at GCP har stabil oppetid for at arbeid skal kunne utføres
Bruke GCP-komponenter	Da GCP er plattformen SVV bruker må prosjektet nødvendigvis basere seg på GCP-komponenter
System som kan implementeres i produksjon	Nødvendig for å svare på oppgaven som stilles
God dokumentasjon underveis	Nødvendig for at teknikere kan reprodusere arbeidet
Deltakerne tilegner seg tilstrekkelig kunnskap underveis	Nødvendig for å greie å løse oppgaven
Regelmessig informasjonsutveksling	Nødvendig for å holde kontroll på prosessen og at ting blir gjort

Figur 6: Suksessfaktorer

5.2 Informasjonsbehov

For å holde interessentene oppdaterte og å ivareta deres behov og krav er det nødvendig å opprettholde god og kontinuerlig informering. Det er derfor viktig at formidlingen av informasjon foregår på en avtalt og forventet måte. Videre er det analysert hvilken informasjon som er nødvendig, til hvem, hvorfor, hvor og når.

Informasjon	Interessent	Formål	Kanal	Tid
Progresjon	SVV	Sikre nødvendig fremdrift	Møter med tilhørende referat, stand-up	Møter hver 2.-3. uke, stand-up to ganger i uken
Tekniske problemer, -utfordringer	SVV	Løse problemer	Slack, stand-up evt. på videomøte	Når problem oppstår
Forstudie	SVV, veileder	Sikre rammer, krav, mål	Teams	Første versjon 01.02.21, sporadisk etter dette
Design-rapport	SVV, veileder	Fastsette design og tekniske spesi-fikasjoner	Teams	Første versjon 22.02.21, sporadisk etter dette
Driftsrapport	SVV, veileder	Dokumentere oppsett av system	Teams	Første versjon 03.05.2021, sporadisk etter dette
Sluttrapport	SVV, veileder	Evaluering av prosjektet	Teams	10.05.21
Prosjekt-håndbok	Veileder	Vise prosess	Teams	Kontinuerlig
Gruppe-dynamikk	Veileder	Ta tak i eventuelle problemer	E-post	Når/hvis nødvendig
Alle rapporter	Veileder, sensor	Vurderings-grunnlag	Inspera	20.05.21
Nyttig lesestoff	Prosjekt-gruppe	Relevant kunnskap	Slack	Sporadisk

Figur 7: Informasjonsbehov

6 Risikoanalyse

En risikoanalyse identifiserer kritiske hendelser, og rangerer disse etter konsekvens og sjansen for at de inntreffer. Hensikten er å redusere risikoen for uønskede hendelser som kan skade prosjektprosessen. Hendelser med spesielt høy risiko behøver strategier for å håndteres. Ofte kan bevissthet rundt hendelsene i seg selv være preventivt, der man kan ta tak i faresignaler for å forebygge risikoen. En godt gjennomført risikoanalyse gir også prosjektmedlemmene en større trygghet rundt prosjektet. Identifikasjon av de mest kritiske hendelsene kommer senere i dette kapittelet.

Analyse av risikoene vises i tabellen under, med sannsynlighetene «Lav», «Moderat», og «Høy», og viktighetene «Ubetydelig», «Tolererbar», «Seriøs» og «Katastrofal»

Kode	Risiko	Sannsynlighet	Viktighet
1	Oppgaven har for stort omfang	Moderat	Tolererbar
2	Sykdom i kritiske perioder	Moderat	Seriøs
3	Produktet fungerer ikke til demo	Lav	Katastrofal
4	Konflikt i gruppa	Lav	Seriøs
5	Problemer med teknologien	Moderat	Seriøs
6	Oppgaver tar lengre tid enn forventet	Høy	Seriøs

Figur 8: Risikoer

De uønskede hendelsene plasseres i en risikotabell for å markere deres alvorlighetsgrad. Hendelsene som havner i grønne celler er forholdsvis milde da de gjerne har lav sannsynlighet og/eller er av liten konsekvens. Disse er det i all hovedsak ikke nødvendig å redusere. Dette betyr derimot ikke at de ikke kan oppstå eller at man ikke bør ha strategier for å håndtere dem.

Hendelsene som plasseres i gule celler har moderat risiko. Disse hendelsene er av større betydning for prosjektet, og kan føre til større problemer om de

finner sted. Disse hendelsene bør ha konkrete strategier for å håndtere.

Hendelsene som plasseres i røde celler er de mest kritiske, og de må reduseres så mye som mulig. Disse hendelsene har både høy sannsynlighet for å inntreffe, og høy konsekvens om dette skjer.

	Tolererbar	**Seriøs**	**Katastrofal**
Lav		4	3
Moderat	1	2, 5	
Høy		6	

Figur 9: Risikoer i risikotabell

Med bakgrunn i disse uønskede hendelsene, har det blitt utarbeidet strategier for å håndtere dem. Disse strategiene presenteres i den underliggende tabellen.

Kode	Risiko	Strategi
1	Oppgaven har for stort omfang	Redusere oppgavens omfang. Prioritere å gjennomføre en mindre del av oppgaven. Dette blir eventuelt diskutert på møte med veileder.
2	Sykdom i kritiske perioder	Fordele arbeidsoppgavene på resten av gruppa. Ved vedvarende helseproblemer må det vurderes å redusere omfanget på oppgaven slik at de gjenværende friske prosjektmedlemmene klarer å gjennomføre prosjektet på egenhånd.
3	Produktet fungerer ikke til demonstrasjon	Finne ut hva som gikk galt og nøste opp i problemet. Forhindre at problemet gjentar seg. Hyppige utrullinger av små endringer gjør at man blir tryggere på endringene og lettere kan finne feilen.
4	Konflikt i gruppa	Bruke gruppekontrakten til å håndtere konflikten.
5	Problemer med teknologien	Lese seg opp, kontakte veileder, be om teknisk hjelp fra Vegvesenet. Eventuelt diskutere bytte av teknologi.
6	Oppgaver tar lengre tid enn forventet	Sette flere folk på oppgaven, eventuelt omprioritere. Se strategi for å håndtere risiko 1.

Figur 10: Strategier for å håndtere risikomomenter

7 Kost/nytte-analyse

Kost/nytte-analysen gjennomføres for å presentere nytten av prosjektet, satt opp mot kostnadene. Dette gir oppdragsgiver et et mer omfattende grunnlag for å kunne beslutte om prosjektet bør gjennomføres eller ikke. En slik analyse ser i første omgang på hvilken effekt prosjektet vil oppnå, også betegnet som *forventet nytte*. Videre går den inn på hva prosjektet vil koste, for så at dette settes opp mot den forventede nytten. Det skilles også mellom *kvantifiserbar* og *ikke-kvantifiserbar* nytte. Den kvantifiserbare nytten er konkret og kan tallfestes, der den ikke-kvantifiserbare nytten er mer abstrakt og vanskelig å tallfeste. Analysen setter altså løsningen slik den er i dag, også kalt *0-alternativet*, opp mot den situasjonen prosjektet resulterer i.

Dette prosjektet er et moderniserings- og digitaliseringprosjekt. Bruk av kunstig intelligens er også noe som egendefineres hos hver enkelt bedrift. Dette fører til at det er vanskelig å tallfeste den kvantifiserbare nytten. Av den grunn vil en kost/nytte-analyse gi begrenset informasjon til oppdragsgiver, sammenlignet med samme analyse i et mer kostnadsrettet prosjekt. Da prosjektet går ut på å utvikle et 'Proof of Concept' tilfører det ikke bedriften en direkte nytte, annet enn å undersøke om et større prosjekt har livets rett. Den indirekte nytten, eksempelvis en mer automatisert veianalyse, er avhengig av det større prosjektet, og av den grunn blir det ikke medregnet. Når det kommer til kostnader, så er det usikkert hva et 'Proof of Concept' vil koste i denne konteksten. Maskinlæringmodellene utvikles utenfor dette prosjektet, og blir ikke medregnet i analysen. Google Cloud Platform benyttes allerede av Statens vegvesen, og inngår i kostnadstaket. Python-programpakker støtter opp Google Cloud Platform i å utvikle produksjonspipelinen har i hovedsak åpen kildekode, og derav minimale kostnader. Prosjektgruppen tar derfor utgangspunkt i kostnadstaket som oppdragsgiver har satt. På bakgrunn av denne begrensede informasjonen, blir det derfor presentert en grovskissert kost/nytte-analyse med redusert omfang.

7.1 Kvantifiserbar og ikke-kvantifiserbar nytte

7.1.1 Kvantifiserbar nytte

Prosjektet er som beskrevet innledningsvis i kapittel 7 et moderniserings- og digitaliseringsprosjekt, fremfor et kostnadsprosjekt. Bedriftens effektmål støtter opp under dette, og består i større grad av ikke-kvantifiserbare mål enn tallfestede. Av denne grunn ble det besluttet å utelate den kvantifiserbare nytten.

7.1.2 Ikke-kvantifiserbar nytte

Da løsningen som utarbeides ikke skal direkte implementeres, men heller bevise en mulighet for implementasjon, skilles det mellom direkte og indirekte ikke-kvantifiserbar nytte. Den direkte nytten gjenspeiler de positive effektene bedriften kan innhente på kort sikt, og som et direkte resultat av prosjektet. Om prosjektet blir vellykket, kan det tas i bruk i Sagaprosjektet. En vellykket implementasjon i et større prosjekt legger til rette for den indirekte nytten. Den kommer ikke som direkte følge av dette prosjektet, men forteller hvordan bedriften kan ta nytte av innføringen av maskinlæringsmodeller i sin helhet og på lang sikt.

7.1.2.1 Direkte nytte

- **Bevise at et større prosjekt er teknologisk levedyktig.** Oppgaven fra bedriften er å utarbeide et 'Proof of Concept' for å foreslå bevis for at idéen er levedyktig. Teknologiprosjekter er svært kostbare, og igangsettelse av slike prosjekter uten et godt grunnlag ender ofte opp som lite vellykket. Et 'Proof of Concept' tilfører altså bedriften nytte ved å foreslå bevis for at et større prosjekt er teknologisk levedyktig.
- **Tilføre bedriften innsikt i tilgjengelige verktøy for prosessering og vedlikehold av maskinlæringsmodeller** . Google Cloud Platform inneholder utallige verktøy for å administrere, prosessere og overvåke maskinlæringsmodeller. Gjennom prosessen vil prosjektgruppen filtrere ut og ta i bruk de mest nyttige verktøyene for produksjonspipelinen, noe bedriften også kan lære av og få innsikt fra i ettertid.

7.1.2.2 Indirekte nytte

- **Modernisere og effektivisere veianalyse**. Prosjektet setter ikke søkelys på en spesifikk maskinlæringsmodell, men det kan tenkes at flere av modellene som tas i bruk av Vegvesenet i fremtiden vil kunne modernisere og effektivisere veianalyse. Dette kan være snakk om maskinlæringsmodeller som analyserer skilt, trafikk, føre, med mer. Om dette prosjektet lykkes, bidrar det til å støtte opp under det større prosjektet som kan føre til dette.
- **Modernisere arbeidsplassen** . Statens vegvesen er stort og består av både kontorarbeidere, veiarbeidere, med mer. Effektive maskinlæringsmodeller vil kunne modernisere og automatisere flere typer arbeidsoppgaver, da spesielt de praktiske. Et eksempel er en effektivisering av vedlikehold av vei. Dette kan videre føre til et redusert behov for arbeidskraft, noe som sparer bedriften for vesentlige ressurser.
- **Sikre arbeidskvalitet hos innleide veientreprenører** . Bedriften leier årlig inn et stort antall entreprenører for å utføre diverse arbeidsoppgaver, da spesielt på vinterstid. Dette omfatter blant annet strøarbeid og måking av snø på veiene. Maskinlæringsmodeller har potensialet til å kunne kontrollere arbeidsmengden hos disse entreprenørene, for å sikre at deres arbeidsoppgaver tilsvarer det som er forventet av daværende vær og veiforhold.

7.2 Bortfall av direkte kostnader

De direkte kostnadene inngår i prosjektets nytteverdi, og omfatter blant annet hva bedriften kan spare inn av kostnader på å innføre det nye systemet. Systemet som skal innføres har verken et eksisterende rammeverk eller programvare, og har derav ingen kostnader som direkte kan økes eller reduseres. Skal man se på bortfall av direkte kostnader må man tenke svært langsiktig, noe som kan oppfattes vagt. Da blir det snakk om bortfall på bakgrunn av en vellykket implementasjon i systemet, og fordi prosjektet ikke retter seg mot en spesifikk maskinlæringsmodell, kan dette omfatte svært

mye. På sikt kan en vellykket implementasjon føre til redusert behov for manuell arbeidskraft, som fører til et bortfall av direkte kostnader. Da dette ikke blir en direkte konsekvens av nåværende prosjektprosess, blir det ikke tatt hensyn til i denne omgangen.

7.3 Estimerte kostnader

Prosjektprosessen har ingen innlysende kostnader, annet enn kostnader tilknyttet ressursbruk i Google Cloud Platform og operasjonelle kostnader senere i forløpet. Da skyløsningen allerede eksisterer i bedriften, er det vanskelig å estimere en eksakt økning i ressurskostnader på bakgrunn av denne prosjektprosessen. Kostnadene er todelte, og man må også beregne arbeidskostnader. Det kan være aktuelt å øke kostnadene på digitale verktøy, om dette kan redusere behovet for manuell arbeidskraft, men det er for tidlig i prosessen til å kunne tallfestes. Av den grunn er det besluttet å ta utgangspunkt i kostnadstaket som ble satt av Statens vegvesen i forkant av forstudien. Prosjektet har et mykt tak på $500 i måneden, dette tilsvarer per januar 2021 sirka 4500 NOK. Dette taket er satt midlertidig for å forhindre store kostnader som følge av uforutsett høy ressursbruk, og kan heves om det skulle være behov.

7.4 Sammenstilling av kost/nytte

Denne delen skal presentere en sammenstilling av verdiene som man kom frem til i kost/nytte-analysen. Da denne forstudien foreløpig ikke kan vise til konkrete verdier innen verken kostnad eller nytte, utelates denne sammenstillingen foreløpig. Dukker det opp en mulighet for tallfesting av kost og/eller nytte senere i prosjektprosessen kan dette avsnittet revideres.

8 Retningslinjer og standarder

8.1 Krav til dokumentasjon

Dokumentasjon som skal produseres i løpet av prosjektet, samt dato for ferdigstillelse, rutiner for godkjennelse og revisjon. Alle i prosjektgruppen er kvalitetsansvarlige, og vil lese igjennom all dokumentasjon uavhengig av hverandre.

- **Forstudierapport:** Undersøker om prosjektet kan gjennomføres innen gitte ressursrammer. Ferdigstilles innen 08.02.2021.
- **Designrapport:** Kvalitetssikrer prosjektprosessen, og viser både design og de tekniske spesifikasjonene som behøves for å få et vellykket resultat. Ferdigstilles innen 22.02.2021.
- **Driftsrapport:** Den endelige prosjektløsningen dokumenteres i denne rapporten. Dette omfatter blant annet teknologivalg og oppsett av løsning. Ferdigstilles innen 03.05.2021.
- **Sluttrapport:** En siste rapport som evaluerer prosjektarbeidet og prosessmålene som ble satt i forstudien. Den skal også gi en oppsummering og oversikt over prosjektet og prosjektresultatet. Ferdigstilles innen 10.05.2021.
- **Prosjekthåndbok:** Utarbeides parallelt med de overstående rapportene, og inneholder timelister, ukesrapporter og oversikt over møtevirksomhet. Ferdigstilles innen 20.05.2021.
- **Vurdering av gruppesamarbeid:** En individuell vurdering av hvor godt prosjektgruppa har samarbeidet. Ferdigstilles innen 20.05.2021.

8.2 Krav til kvalitetsgjennomganger

Dokumentene skal kontrolleres og revideres av både prosjektdeltakerne og de øvrige interessentene, dette gjelder spesielt de større dokumentene som forstudierapport, designrapport, driftsrapport og sluttrapport. Dokumentasjon fra møtevirksomhet er mindre formelt og behøver ikke samme nivå av kvalitetskontroll. Det stilles også større språklig krav til

hoveddokumentasjonen. Alle interessenter vil likevel få tilbud om å revidere og gi tilbakemelding på all dokumentasjon. For øvrig vil rapporter, tidsbruk og fremdrift rapporteres til, og vurderes av, intern sensor og veileder Jostein Lund, i samarbeid med ekstern veileder Torgeir Thoresen.

Endelig resultat vil vurderes og karaktersettes av Jostein Lund fra NTNU, sammen med ekstern sensor Safurudin Mahic fra Bekk på vegne av Statens vegvesen. Vurderingen stilles på bakgrunn av dokumentkvalitet og prosjektløsning. Det vektlegges også at all dokumentasjon blir levert i henhold til tidsfristen 20.05.2021, og at prosjektperioden avsluttes med en godt gjennomført presentasjon.

8.3 Krav til standarder og metoder

Prosjektgruppen skal bruke utvalgte verktøy i løpet av prosjektperioden:

- All formell dokumentasjon og rapportskriving skal skje i LaTeX, i samskrivingsprogrammet Overleaf.
- All lagring av formell prosjektrelatert dokumentasjon og fagstoff skjer i felles Microsoft Teams-gruppe, der alle interessenter har tilgang. Her foregår også oppfølging av prosjektprosessen.
- All uformell dokumentasjon og kladder utarbeides i en felles Microsoft OneNote.
- Møteinnkalling skal skje via kalenderfunksjon i Outlook/Teams, da dette plasserer møtet direkte i mottakers kalender.

8.4 Endringshåndtering

Endringsønsker kan komme på vegne av bedrift, prosjektdeltakerne, brukere eller andre interessenter, og om det oppstår behov for endring skal dette diskuteres med og godkjennes av bedrift snarest. Endringsønsker skal behandles formelt og forretningsmessig, dette gjelder større endringer som prosjektets omfang og teknologivalg.

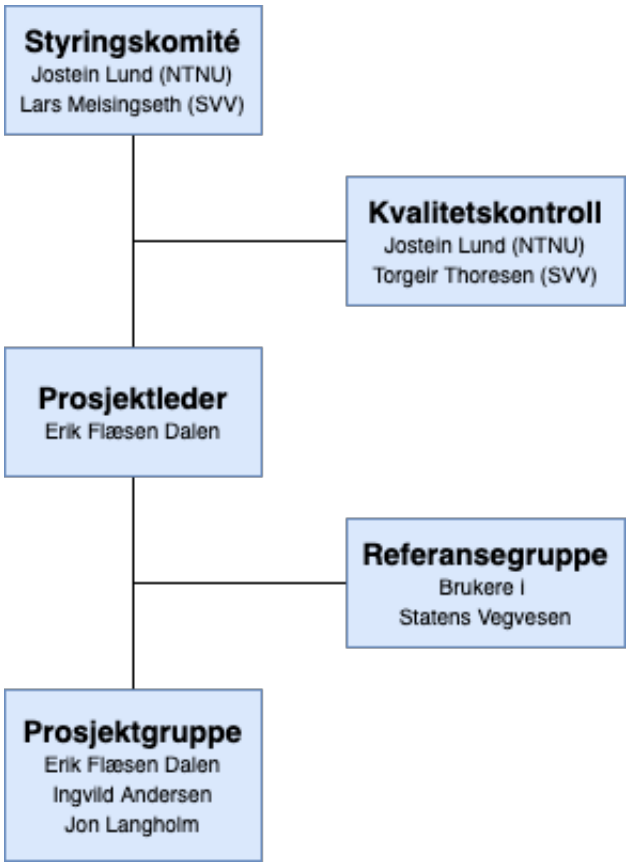
Framgangsmåten for håndtering av endringsønsker er:

1. Dokumenter endringens innhold
2. Analyser konsekvensene for prosjektet
3. Beregn eventuell kost/nytte
4. Godkjennelse og aksept
5. Logg endringen
6. Juster planene
7. Informer interessentene
8. Gjennomfør endringen

Underveis i programvareutviklingen følger prosjektet Scrum for endringshåndtering, og ikke den overstående listen.

9 Prosjektorganisering

Prosjektet følger en klassisk prosjektorganisering, bestående av prosjektleder, styringskomité, kvalitetskontroll, prosjektgruppe og referansegruppe. Følgende organisering er valgt da det tyder på å passe best i dette prosjektet.



Figur 11: Prosjektorganisering

Prosjektleder

Erik Flæsen Dalen er utnevnt av prosjektgruppen til å inneha den formelle prosjektlederrollen gjennom prosjektet. Prosjektleders ansvar er i dette tilfellet å legge til rette og sørge for god kommunikasjon mot prosjektets interessenter og øvrige deltakere, løpende kvalitetskontroll av dokumentering og løsninger, overholdelse av tidsfrister og prosesskontroll.

Styringskomité

Definert til å være prosjektets oppdragsgiver, støttet av faglig kontakt, består

styringskomitéen av Lars Meisingseth v/Statens vegvesen og Jostein Lund v/NTNU. Lars Meisingseth fungerer som prosjektets eier og har beslutningsmyndighet for prosjektet.

Kvalitetskontroll

Som et bindeledd mellom prosjektleder og styringskomite fungerer kvalitetskontroll som kontrollør av prosjektets fremdrift, i form av dokumentering og prosess, sett fra NTNUs side. I tillegg skal også kvalitet på utført arbeid kontrolleres for å sørge for best mulig måloppnåelse.

Referansegruppe

Representanter for ansatte i Statens vegvesen, i hovedsak, men ikke begrenset til, Safurudin Mahic og Torgeir Thoresen, innehar rollen som referansegruppe. De representerer brukerne av systemet, og vil inneha en liknende rolle som kvalitetskontroll, men i dette tilfellet fra oppdragsgivers perspektiv. Teknisk kvalitetskontroll og beslutningsstøtte er typiske oppgaver.

Prosjektgruppe

Prosjektgruppen inneholder prosjektets utførende, nevnelig Erik Flæsen Dalen, Ingvild Andersen og Jon Langholm. Deres ansvar er å sørge for at prosjektet leverer et kvalitetsfylt resultat og dekker og oppnår de fastsatte krav og mål. Prosjektgruppen går inn i prosjektet med flytende arbeidsoppgaver, men forhåndsdefinerte fokusområder. Erik tiltrer rollen som prosjektleder, Ingvild fokuserer på dokumentering og rapporter, mens Jon fokuserer på utvikling og tekniske løsninger. Deltakerne bidrar likevel i de forskjellige delene etter behov.

10 Anbefaling om videre arbeid

Statens vegvesen ønsker seg altså en maskinlæringspipeline som kan brukes til forskjellige prosjekter i fremtiden. Bachelorgruppen har i forstudiet undersøkt mulige løsninger og teknologier som kan nyttes for å skape et 'Proof of Concept'. Det var vanskelig å tallfeste kostnad og nytteverdi, men den potensielle nytteverdien Vegvesenet kan hente i det lange løp overskrider det man antar ender opp som kostnad. Det er derfor vanskelig å argumentere for at prosjektet ikke bør gjennomføres. For deltakerne selv, vil kunnskapen og erfaringen som opparbeides under gjennomføringen av prosjektet være til høy verdi. Bachelorgruppen er høyst motiverte til å gå videre med prosjektet og dette anbefales på det sterkeste.

Referanser

[1] Regjeringen: *Statens vegvesen*, 2021.
Hentet fra:
`https://www.regjeringen.no/no/dep/sd/org/underliggende-etater/statens-vegvesen/id443412/`.
Lastet ned: 15. januar 2021

[2] J. A. Langholm, I. Andersen og E. F. Dalen, "Prosjekthåndbok", *NTNU*, Mai 2021. [Vedlegg].
Hentet fra: Vedlegg i innlevering av oppgave.

2 Designrapport

MLOps på Google Cloud Platform
Designrapport

Ingvild Andersen, Jon Akselberg Langholm, Erik Flæsen Dalen

19. mai 2021

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
11.03.2021	1.0	Førsteutkast	Erik F.D., Ingvild A. og Jon L.
14.03.2021	1.1	Mindre revidering etter tilbakemeldinger fra veileder	Erik F.D., Ingvild A. og Jon A. L.
19.05.2021	1.2	Språkvask	Erik F.D., Ingvild A. og Jon A. L.

Innhold

1 Innledning **5**
1.1 Dokumentets hensikt 5
1.2 Innhold 5
1.3 Avgrensning 6
1.4 Definisjoner og forkortelser 6

2 Kort om kunden og behov **7**

3 Hvorfor valgt teknologi **8**

4 Tekniske løsninger **9**
4.1 Introduksjon til teknologi 10
4.1.1 MLOps 10
4.1.2 Plattform: Google Cloud Platform (GCP) 12
4.1.3 Rammeverk: TensorFlow Extended (TFX) 14
4.1.4 Orkestrering: Kubeflow Pipelines (KFP) 16
4.2 Verktøy og ressurser 18
4.2.1 Prosjekter i Google Cloud Platform 18
4.2.2 Utviklingsmiljø 20
4.2.3 Akseptanse- og testmiljø 20
4.2.4 Produksjonsmiljø 20
4.3 Detaljerte løsningsbeskrivelser 21
4.3.1 TensorFlow Extended (TFX) 21
4.3.2 Google Cloud Platform (GCP) 22
4.3.3 Kubeflow Pipelines (KFP) 23
4.3.4 Flyt i pipelinen 24
4.3.5 Forutsetninger og avhengigheter 26
4.3.6 Programvare og systemkrav 26
4.3.7 Oppdatert GANTT-diagram 26

5 Alternative flyter i GCP **27**

Referanser **32**

Figurer

2 Overordnet flyskjema for TFX-pipeline. Inspirert av TensorFlow-dokumentasjon.[5] 16
3 Komponentspesifikk eksempelflyt i TFX-pipeline på GCP 16
4 Kubeflow Pipelines integrert med Google Cloud Platforms ML-verktøy 18
5 TFX-komponentene satt sammen i Kubeflow 24
6 Faktisk arbeidsprosess. 27
7 Planlagt arbeidsprosess. 27

1 Innledning

1.1 Dokumentets hensikt

Følgende dokument er en designrapport utarbeidet i sammenheng med bacheloroppgaven 'MLOps på Google Cloud Platform' på oppdrag fra Statens vegvesen. Dokumentets hensikt er å tilby leseren en konseptuell og tekstlig beskrivelse av løsningen oppgaven skal resultere i. Dokumentet er en del av prosjektprosessens kvalitetssikring, og omfatter derfor en innsikt i bedriftens krav og behov, samt en oversikt over foreslått løsningsdesign for å møte disse. Denne beskrivelsen vil sikre gjensidig kommunikasjon mellom oppdragsgiver og oppdragstaker, slik at alle parter sitter på samme mengde informasjon når dokumentet er ferdigstilt. Målet er at prosjektgruppen og Statens vegvesen skal bli samstemte i valg av løsningsdesign, tekniske detaljer og forventninger. Designrapporten bygger videre på beslutningene som ble tatt i forstudierapporten. Dokumentet begrenses til å beskrive forslag til løsning av prosjektoppgaven, men ikke hvordan man kan reprodusere denne løsningen. En slik innføring vil finne sted i driftsrapporten.[34] Forslag til løsning vil også innebære en innføring i valgt teknologi og hvordan den samsvarer med oppgavekrav fra Statens vegvesen.

1.2 Innhold

Første kapittel i dokumentet beskriver dokumentets hensikt (1.1) og en oversikt over dets innhold (1.2). Videre presenteres en kort oversikt over dokumentet og oppgavens avgrensning (1.3). Definisjoner og forkortelser befinner seg i ekstern ordbok.[32] Kapittel 2 går overordnet inn på kunden og behov, altså hvem kunden er og hva de ønsker ut av prosjektprosessen. Kapittel 3 og 4 går inn på bakgrunnen for valg av teknologi, samt en oversikt over disse. Helt konkret innebærer det overordnet innføring i MLOps (4.1.1), Google Cloud Platform (4.1.2), TensorFlow Extended (4.1.3) og Kubeflow Pipelines (4.1.4). Videre i kapittel 4 får leseren en innføring i prosjektets verktøy og ressurser, da spesielt en beskrivelse av prosjekter i Google Cloud Platform (4.2.1), samt en gjennomgang av utviklingsmiljø (4.2.2), akseptanse- og testmiljø (4.2.3) og produksjonsmiljø (4.2.4). I avsnitt 4.3 gjennomgås

detaljerte løsningsbeskrivelser, TFX (4.3.1), GCP (4.3.2), KFP (4.3.3), flyt i pipelinen (4.3.4) forutsetninger og avhengigheter (4.3.5), programvare og systemkrav (4.3.6) før oppdatert Gantt-diagram (4.3.7). Til slutt i rapporten finnes en overordnet oversikt over alternative flyter i GCP (5) før en liste over referanser og vedlegg.

1.3 Avgrensning

Som beskrevet i forstudierapporten[25] går prosjektets resultatmål ut på å ferdigstille et 'Proof of Concept' på en produksjonspipeline for en maskinlæringsmodell i Google Cloud Platform. Dette omfatter å kunne kjøre ut maskinlæringsmodeller i Google Cloud Platform, samt nye versjoner av disse. Det handler også om å kunne bruke maskinlæringsmodellene til å hente ut prediksjoner, overvåke deres prediksjonskvalitet, trening og skalering. Løsningen skal være kompatibel med Google Cloud Platform, men prosjektgruppen står fritt til å vurdere andelen selvskreven kode opp i mot eksisterende grensesnitt i GCP. Det tas forbehold om at noe funksjonalitet kan sløyfes ettersom oppgavens omfang spesifiseres videre.

Prosjektgruppen skal ikke utvikle maskinlæringsmodellene, drifte eller implementere produksjonspipeline, drive opplæring, sette opp miljø i Google Cloud Platform, eller legge til funksjonalitet som ikke er etterspurt. Dette dokumentet skal presentere og begrunne et forslag til løsningsdesign, men omfatter ikke en gjennomgang av denne løsningen. Sistnevnte kommer i driftsrapport.[34] Det må også nevnes at løsningsforslaget ikke testes i sin helhet før arbeidet med driftsdokumentasjon, noe som kan føre til endringer i valgt løsning. Om dette skulle skje vil det kommuniseres tett med oppdragsgiver, og det vil beskrives i sluttrapporten.[35]

1.4 Definisjoner og forkortelser

Det er utarbeidet en egen ordbok for prosjektet, som kan finnes i vedlagt ordbok.[32]

2 Kort om kunden og behov

Oppdragsgiver er dataplattformen Saga, på vegne av Statens vegvesen. Organisasjonen har ansvaret for Norges riksveier, noe som innebærer forvaltning, utredning, planlegging, bygging, drift og vedlikehold av disse. Statens vegvesen har gjennom Saga en visjon om å gjøre organisasjonen mer datadrevet. Hensikten er å bruke data til å oppdage nye sammenhenger og muligheter, ta mer faktabaserte beslutninger, samt utnytte organisasjonens store datamengder på en mer hensiktsfull måte. Dagens løsning preges av uheldige formater og lite tilfredsstillende API-er. Deling av data og tversanalyser vektlegges også i liten grad.[1]

Saga ønsker at prosjektgruppen skal utforske og utvikle et 'Proof of Concept', i form av en produksjonspipeline som kjører i Google Cloud Platform. Denne løsningen skal kunne ta inn og bytte ut maskinlæringsmodeller, mate disse med ny inndata, gjennomføre trening, være skalerbar, hente ut prediksjoner, samt overvåke kvaliteten på disse. Produksjonspipelinen skal være mest mulig automatisert, og skal kreve minst mulig interaksjon fra datavitere. En ønskesituasjon er at man kan sette maskinlæringsmodeller ut i produksjon, og at trening, testing, overvåking og serving av prediksjoner er tilnærmet selvgående. Dette vil dekke de behov SVV ønsker for å kunne ta i bruk dataen de sitter på så effektivt som mulig. Oppgaven skal løses ved bruk av Google Cloud Platform, og underliggende verktøy er til disposisjon. Prosjektgruppen står likevel fritt til å utforske andre alternativer, så lenge løsningen fungerer godt med plattformen. Oppdragsgiver er fortsatt i utforskningsfasen når det kommer til maskinlæring, og ønsker derfor at løsningen skal inneholde en sammenligning av løsninger, og en forklaring på hvorfor gjeldende produkt-teknologi-løsning ble valgt. Bakgrunnen for dette er at de vil finne teknologien som passer best for deres drift og bruk.

3 Hvorfor valgt teknologi

Valg av plattform ble som tidligere nevnt Google Cloud Platform. Dette valget ble innlysende da Statens vegvesen allerede bruker denne skyløsningen, og å samle alle tjenester på ett sted forenkler utvikling, drift og vedlikehold betraktelig. GCP er også en skyplattform som tilbyr flere verktøy for maskinlæring, noe som gir en stor mengde muligheter når man skal angripe maskinlæringsoppgaver. Plattformen er også langt framme når det kommer til maskinlæring, sammenlignet med lignende plattformer på markedet. Videre ble det besluttet å ta i bruk skalerbar teknologi som kunne integreres sømløst med GCP, og som spesielt kunne prosessere maskinlæringsmodeller basert på TensorFlow. Bakgrunnen for at prosjektgruppen har valgt å fokusere på kun maskinlæringsmodeller basert på TensorFlow, er at TensorFlow er det mest utbredte rammeverket for utvikling av maskinlæringsmodeller.[31] Det ble besluttet å ta i bruk TensorFlow Extended (TFX) som rammeverk og Kubeflow Pipelines som orkestrator, da disse teknologiene er godt tilpasset slike modeller. TFX er de individuelle komponentene i pipelinen, og Kubeflow Pipelines passer på at disse kjøres i riktig rekkefølge. Disse tjenestene er også skalerbare ende-til-ende tjenester som kan utvides og egendefineres etter bedriftens ønske i senere tid. Blant annet kan Kubeflow Pipelines nyttes som orkestrator for maskinlæringsmodeller basert på andre teknologier enn TensorFlow.

Rammeverk og orkestrator er begge tjenester som er åpen for fri bruk og vil ikke påløpe kostnader ved bruk av disse. Pipelinen vil kjøre integrert med GCP, og det vil derav påløpe kostnader tilknyttet bruk av GCPs tjenester. Da prosjektet fortsatt er i en eksperimentell fase er det vanskelig å gi eksakte tall. For øvrig kan bedriften senere tilpasse selv hvor mye de vil integrere pipelinen med GCP. En tett integrasjon vil føre til høyere kostnader, men skyplattformen har også avanserte maskinlæringsverktøy som kan sikre kvalitet i trening, overvåking og levering av prediksjoner. Disse verktøyene er også svært avanserte, noe som gjør dem vanskelig å gjenskape på egen hånd utenfor plattformen.

4 Tekniske løsninger

Valg av løsning ble gjort på bakgrunn av krav fra Statens vegvesen og omfang av prosjektet. Det ble tidlig i planleggingen av løsningsdesign besluttet at teknologivalg skulle implementere prinsippene bak MLOps, som er maskinlæringens svar på DevOps.[32] Bakgrunnen var at løsningen skulle være en automatisk trigget pipeline, der de som utvikler maskinlæringsmodeller, og de som har dem i produksjon, skal kunne samarbeide uten problemer. Dette gjelder helt fra modellene påbegynnes til de kjøres ut i produksjon og leverer prediksjoner. MLOps reflekterer dette ved å legge opp til automatisering hele veien fra man setter modeller ut i produksjon, til man leverer prediksjoner. Videre ble det besluttet å ta i bruk Google Cloud Platform for å lagre og kjøre løsningen og dens underliggende data, da dette er en skyplattform Statens vegvesen bruker i det daglige og er godt kjent med.

Når man skal opprette en automatisert maskinlæringspipeline behøver man ikke bare komponenter, men også et rammeverk og en orkestrator. Rammeverket konfigurerer pipelinen, og orkestratoren kontrollerer kjøringen av denne. Det vil si at orkestratoren styrer og dirigerer rekkefølgen på komponentene og arbeidet pipelinen består av. Det finnes flere rammeverk å velge i når man skal sette sammen maskinlæringspipeliner, og det ble i dette prosjektet besluttet å gå videre med TensorFlow Extended (TFX). TensorFlow er svært utbredt innen utvikling av modeller i dag, og det er derfor ønskelig med en pipeline som best mulig kan tilpasses disse. TFX er et ende-til-ende verktøy, og rammeverket støtter opp om alt fra førprosessering av data til trening og levering av produksjonsklare modeller. Rammeverket implementerer også prinsippene bak MLOps, og derav ønsket funksjonalitet og krav fra oppdragsgiver. Som orkestrator skal prosjektet ta i bruk Kubeflow Pipelines, som er en plattform for å bygge og kjøre ut portable og skalerbare maskinlæringspipeliner basert på Docker-konteinere. Løsningen tilbyr en ende-til-ende orkestrering som legger til rette for gjenbruk av kode, pipeliner og komponenter, noe som lønner seg for Statens vegvesen, som ønsker en mulighet for rask skalering.

4.1 Introduksjon til teknologi

Dette avsnittet skal overordnet beskrive de mest aktuelle verktøyene for å løse oppgavebeskrivelsen. Det er ikke en oversikt over valgt løsning, men heller en mulighet for leser å sette seg inn i de ulike verktøyenes funksjonalitet. Det er fordelaktig at leser kjenner til den viktigste funksjonaliteten som er aktuell, da det senere i rapporten skal argumenteres for valg av endelig løsning. Statens vegvesen er relativt ferske innen maskinlæring, og ønsker derfor en oversikt over mulige løsninger, i tillegg til en beskrivelse av og begrunnelse for valgt løsning. Sistnevnte gjennomgås i avsnittet detaljert løsningsbeskrivelse".

4.1.1 MLOps

Følgende informasjon som beskriver MLOps er i store deler basert på dokumentasjon fra Google Cloud Platform. Dette omfatter i hovedsak løsningsartiklene *Architecture for MLOps using TFX, Kubeflow Pipelines, and Cloud Build*[5] og *MLOps: Continuous delivery and automation pipelines in machine learning*.[2]

MLOps og DevOps baserer seg begge på konseptene rundt kontinuerlig integrasjon (CI) og kontinuerlig levering (CD), men CI innen MLOps handler ikke kun om testing og validering av kode, men også av data, dataskjema og modeller. CD omfatter ikke kun levering av en enkelt programvarepakke, men hele systemer og pipeliner. MLOps presenterer også et nytt konsept kalt kontinuerlig trening (CT), som er unikt innen maskinlæringssystemer. Dette beskriver en automatisk og iterativ om-trening og levering av modeller. Metoden er mye mer omfattende enn DevOps innen testing, som omfatter datavalidering, samt en kontinuerlig evaluering av modellenes kvalitet. Utkjøring kan også være utfordrende, da modeller behøver en produksjonspipeline som trener og kjører ut nye versjoner. Dårlige ytelse i maskinlæringssystemer skyldes som regel endringer i dataprofiler, i motsetning til lite optimal kode hos programvaresystemer. Maskinlæringsmodeller må konstant overvåkes, og ha klare tiltak ved endringer i data eller en reduksjon i prediksjonskvalitet.[2]

MLOps handler altså om å forenkle samarbeidet mellom de som utvikler maskinlæringssystemer, og de som vedlikeholder dem. Praksisen vektlegger

en størst mulig grad av automatisering når modeller skal ut i produksjon. Hos bedrifter som planlegger få maskinlæringsmodeller med et lavt vedlikeholdsbehov, så kan en produksjonspipeline med manuelle triggere for trening, validering og utkjøring holde mål. Dette gjelder ikke Statens vegvesen, da det er en stor organisasjon som satser tungt på digitalisering, maskinlæring og automatisering av arbeidsoppgaver. I deres tilfelle vil det lønne seg å innføre MLOps først som sist, noe som vil omfatte at hele pipelinen består av automatiske triggere. Her skal alt fra innførsel av data til levering av prediksjoner skje så sømløst som mulig.

Levere en modell til produksjon

Stegene fra utvikling til overvåkning av en maskinlæringsmodell kan overordnet beskrives som vist under. Når modellen utarbeides for første gang handler de første punktene også om å skape selve modellen, når modellen settes i produksjon kan disse tolkes som å hente inn ny data, analysere datatype og forberede disse for modellen.

1. **Hente ut data**
   Velge relevant data fra diverse datakilder.
2. **Analysere data**
   Undersøkende dataanalyse utført av dataviter. Handler om å forstå tilgjengelig data som behøves for å bygge maskinlæringsmodellen.
3. **Forberede data**
   Her formatteres og renses data, samt splittes inn i sett for trening, validering og testing.
4. **Trening av modellen**
   Her implementeres i første omgang forskjellige algoritmer som trenes på forberedt data, samt en innstilling av hyperparametere. Dette er parametere som kontrollerer maskinlæringsmodellens læringsprosess. De påvirker ikke læringen i seg selv, men kvaliteten på prosessen rundt. Disse varierer for eksempel ut ifra hvilken algoritme dataviteren velger å basere modellen på. Resultatet er en ferdigtrent modell.
5. **Evaluering av modellen**
   Modellen evalueres på et sett av data den ikke har blitt eksponert for

tidligere, også kalt test-sett. Utdata fra denne fasen er metrikker som måler modellens prestasjon.

6. **Validering av modellen**
   Om modellen bekreftes god nok for utkjøring, og dens prediksjoner er bedre enn grunnlinja (baseline), eller tidligere godkjente modell, blir den godkjent for levering.

7. **Levering av modellen**
   Den validerte modellen kjøres ut til et miljø for å levere prediksjoner. Denne utkjøringen kan enten være som en mikrotjeneste med et REST API for å levere nettbaserte prediksjoner, en innebygget modell til en mobil enhet, eller som en del av et system som henter ut prediksjoner via terminal.

8. **Overvåkning av modellen**
   Når modellen er kjørt ut i produksjonsmiljø, blir den kontinuerlig overvåket. Om evnen til korrekte prediksjoner reduseres, kan det igangsettes en ny iterasjon av hele prosessen.

Videre gjennomgang av eksisterende teknologiløsninger vil så nært som mulig gjenspeile denne prosessen.

4.1.2 Plattform: Google Cloud Platform (GCP)

Google Cloud Platform er en skytjeneste som leveres av Google. Plattformen tilbyr et stort antall datatjenester som blant annet datalagring, dataanalyse og maskinlæring. Den tilbyr både IaaS, PaaS og 'serverless computing'. Statens vegvesen er allerede kjent med GCP, noe som gjorde det naturlig at bedriften ønsket en løsning som er integrerbar med denne. Utviklings-, akseptansetest- og produksjonsmiljø er av den grunn opprettet her. Sett bort i fra bedriftens krav, så er Google Cloud Platform fortsatt et solid valg av maskinlæringsplattform. GCP tilbyr et stort antall maskinlæringsverktøy, og man skal teoretisk sett kunne trene, validere og levere modeller uten å bruke programmeringskode. Plattformen tilbyr lagring av data under alle steg av pipelinen, samt trening direkte på plattformen gjennom tjenesten AI Platform. Om utviklere ønsker å lage pipelinen selv, men med egen kode, så kan dette

også kjøres i GCP, og man har valget om hvor mange av plattformens egne tjenester som skal blandes inn.[16] Dette legger opp til at pipeliner kan egendefineres, og tilpasses bedriften og maskinlæringsalgoitmene best mulig.

Google Cloud Platform tilbyr et stort antall tjenester for maskinlæring. I dette prosjektet skal man ikke nødvendigvis ta i bruk alle nevnte tjenester, men siden GCP er så tett integrerbart med denne løsningen, er det fordelaktig at bedriften har en overordnet oversikt over mulighetene.[20]

- **Cloud AutoML**
  En tjeneste som lar brukerne automatisk trene, evaluere, forbedre og kjøre ut egendefinerte maskinlæringsmodeller direkte i Google Cloud. Her benyttes kun et grafisk grensesnitt. Om brukeren ønsker det kan man også koble egendefinert kode opp mot AutoML.[17]

- **AI Platform**
  Her kan utviklere av maskinlæringsmodeller ta deres modeller fra utvikling til produksjon og levering. AI Platform er integrert med blant annet BigQuery og Data Labeling Service (DLS). BigQuery hjelper med lagring av store mengder data, og DLS bistår med å merke og forberede data for trening og testing av modellen. AI Platform kan også integreres med pipeliner man lager selv, da spesielt gjennom Kubeflow Pipelines og TFX-komponenter.[18] Dette gjennomgås nærmere i neste avsnitt.

- **AI Hub**
  Google Cloud tilbyr hver enkelt bruker en egen katalog for å laste opp og eksperimentere med maskinlæringsmodeller, AI-komponenter og ende-til-ende pipeliner. Her kan man dele ML-pipeliner, notisbøker, modeller og annet ML-innhold.[19]

- **BigQuery ML**
  Lar deg opprette og kjøre maskinlæringsmodeller i BigQuery gjennom bruk av standard SQL-spørringer. Lar altså SQL-brukere bygge modeller ved hjelp av eksisterene SQL-kunnskap. Nyttig verktøy spesielt for datavitere som vil evaluere modeller i BigQuery, uten at de har mye kunnskap om maskinlæringrammeverk i forkant.[21]

- **Cloud Storage Buckets**
  Ikke et maskinlæringsverktøy i seg selv, men svært sentralt for å kunne sette modeller i produksjon. Når man skal behandle en maskinlæringsmodell må modellen, data og tilhørende artefakter ha en lagringsplass. Dette skjer via Cloud Storage Buckets. Her oppretter enten utvikleren en bucket manuelt, eller kjører kode som oppretter dem automatisk. Disse kan nås innad hvert enkelt prosjekt, om ikke man tildeler spesielle tillatelser. Cloud Storage behøves blant annet for å lagre modelldata som treningsapplikasjon, modell, artefakter, inndata (treningsdata) og utdata.[22]

Dette er bare et lite utvalg av hva som finnes av maskinlæringsverktøy i GCP. Det finnes også andre tjenester som kan være nyttig, som Dataflow[23] og Dataproc,[24] men disse gjennomgås ikke i dybden per nå.

4.1.3 Rammeverk: TensorFlow Extended (TFX)

TensorFlow Extended (TFX) tilbyr et rammeverk for å opprette pipeliner bestående av TFX-komponenter. Dette inkluderer delte biblioteker for integrering, som behøves for å kjøre og overvåke TensorFlow maskinlæringsmodeller.[3] Dette rammeverket er et ende-til-ende-verktøy som støtter opp under alt fra transformasjon av data til trening og levering av produksjonsklare modeller.[4] TFX implementerer prinsippene bak MLOps, og de forskjellige komponent tilbyr lagring, konfigurering og orkestrering av maskinlæringsmodeller. TFX-pipeliner er skalerbare og takler tunge maskinlæringsoppgaver, som modellering, trening, og validering. TFX-pipeliner takler også levering og administrering av utkjøringer. TFX-pipeliner kan orkestreres av Apache Airflow og Kubeflow Pipelines.

TensorFlow Extended har flere underliggende biblioteker som støtter opp under ønsket pipeline-funksjonalitet, slik som trening og utkjøring, samt integrasjon med GCP. Videre vil fasene i MLOps listes opp, sammen med TFX-komponenter som løser de forskjellige fasene, og tjenester disse integreres med i Google Cloud Platform.

Eksempelpipeline basert på TFX:
Informasjon er hentet fra Google Cloud Platforms dokumentasjon.[5]

1. **Hente ut data**
   I første steg høstes data, og det kommer datafiler ut som brukes til trening og evaluering av modellen.

2. **Validere data**
   Her kan man bruke TensorFlow-biblioteket *TF Data Validation (TFDV)*[6] for å detektere dataanormaliteter. Her valideres ny data mot dataskjema, og om det oppdages avvik må enten skjema eller data oppdateres. TFDV fungerer godt på GCP gjennom Dataflow.
   Kan integreres med Google Cloud: Dataflow.

3. **Transformasjon av data (førprosessering)**
   Her kan biblioteket *TF Transform (TFT)*[7] tas i bruk. Når inndata er ferdig validert, splittes det som kjent opp i test- og treningssett for maskinlæringsmodellen, noe TFT kan bidra med. Systemet sitter igjen med filer for å trene og evaluere modellen, vanligvis i *TFRecords* format. Det opprettes også transformasjonsartifakter som bidrar med å eksportere den lagrede modellen etter trening.
   Kan integreres med Google Cloud: Dataflow.

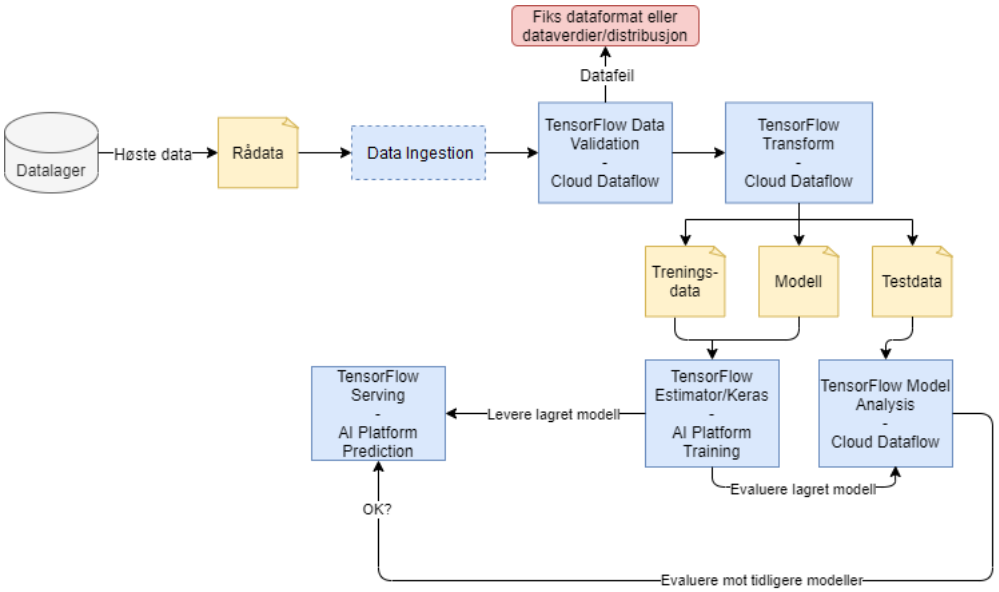
4. **Trening og tilpassing av modell**
   For å implementere og trene maskinlæringsmodellen kan *TF Estimators og Keras*[8] tas i bruk. Disse tilbyr API-er som kan ta i bruk utdata fra punkt 3 (transformasjon). *Keras tuner* kan tilpasse hyperparametere, eller man kan bruke andre tjenester som *Katib*.[33] Dette steget ender med en lagret modell som brukes til evaluering, og en annen som brukes til levering av modellen og prediksjoner til et grensesnitt.
   Kan integreres med Google Cloud: AI Platform: Jobs.

5. **Evaluering og validering av modell**
   Modellen evalueres på et test-datasett for å måle dens prediksjonskvalitet. Her kan man bruke *TF Model Analysis (TFMA)*,[9] som evaluerer modellkvaliteten i sin helhet, og identifiserer hvilke deler av modellen som yter dårlig. Denne delen bestemmer om modellen skal settes ut i produksjon eller ikke.
   Kan integreres med Google Cloud: Dataflow.

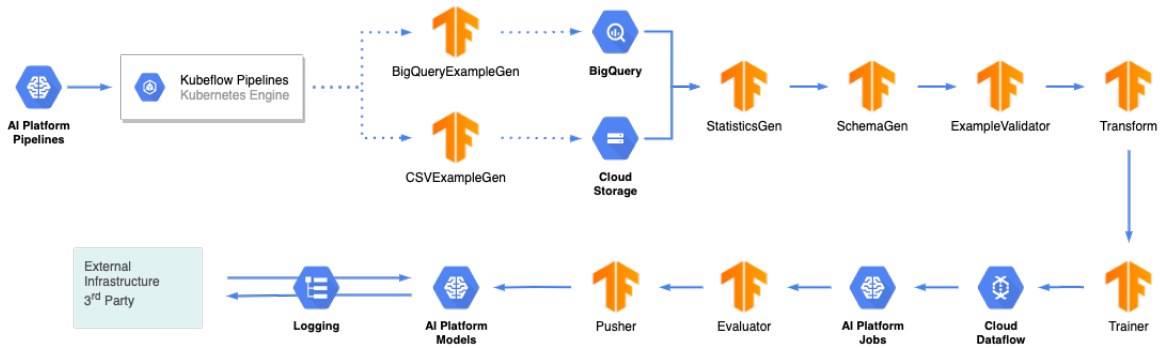
6. **Levering av modell for prediksjoner**

Når modellen er validert, kan den leveres som en mikrotjeneste for å gi nettbaserte prediksjoner via biblioteket *TF Serving (TFS)*.[10] Utdata blir en prediksjon fra den trente maskinlæringsmodellen. Eventuelt kan man lagre den trente modellen i et modellregister.

Kan integreres med Google Cloud: AI Platform: Models.



Figur 2: Overordnet flyskjema for TFX-pipeline. Inspirert av TensorFlow-dokumentasjon.[5]



Figur 3: Komponentspesifikk eksempelflyt i TFX-pipeline på GCP

4.1.4 Orkestrering: Kubeflow Pipelines (KFP)

En orkestrator behøves for å koble sammen de ulike komponentene i produksjonspipelinen. Den sørger for at pipelinen går automatisk i ønsket sekvens, etter planlagte triggere[11] slik at neste komponent i pipelinen får

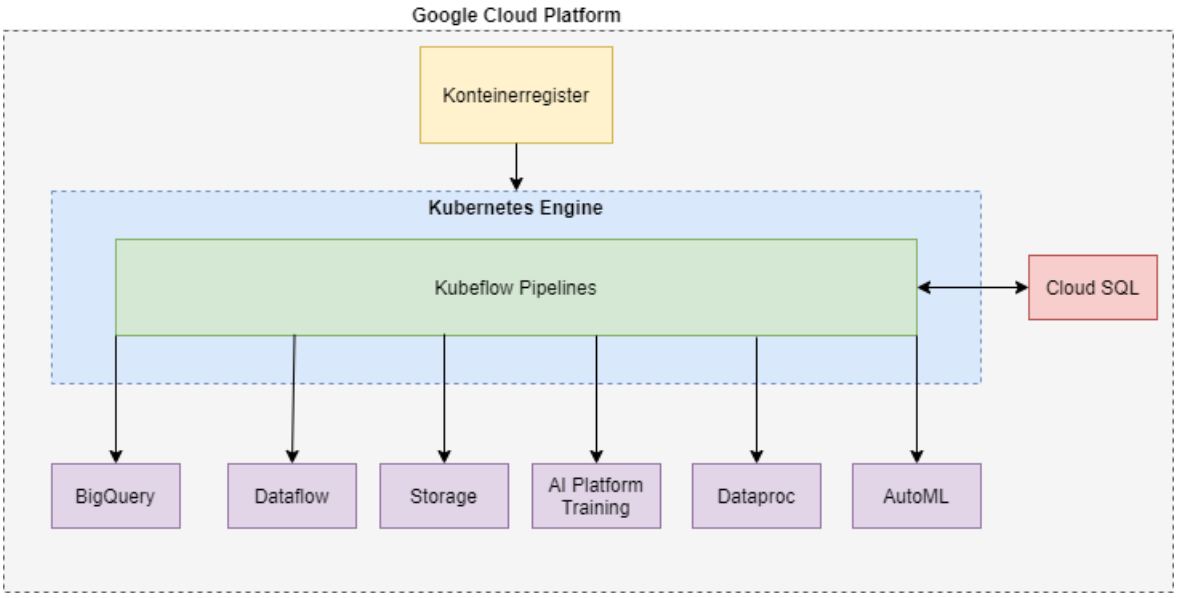
nødvendig inn-data fra foregående komponent. Kubeflow er et rammeverk som stammer fra konteiner-orkestratoren Kubernetes, og det består av flere tjenester for maskinlæring. En av disse tjenestene kalles for Kubeflow Pipelines (KFP). KFP lar deg sette sammen, orkestrere og automatisere maskinlæringssystemer. Hver komponent kan kjøre på Kubeflow eller Google Cloud Platform. Målet er å skape en ende-til-ende orkestrering, samt legge til rette for å enkelt kunne gjenbruke kode og pipeliner. KFP består av en 'engine' for planlegging av hvordan pipelinene skal kjøres. Her brukes Argo Workflows[12] for å orkestrere Kubernetes-ressursene. Videre innehar KFP en Python SDK[13] for å definere og manipulere komponentene. Orkestrereren kan også samhandle med notebooks som Jupyter, og den har en metadata-lagring for informasjom om kjøring, modeller, datasett og andre artefakter. Kubeflow Pipelines tilbyr også flere definerte komponenter som kan kjøre tjenester hos Google Cloud Platform. Disse komponentene bidrar til at man kan utføre oppgaver gjennom tjenester som BigQuery, Dataflow, Dataproc og AI Platform.

Oppbygging

- **Komponenter**
  Kode pakket som Docker-avbildninger.[14] En enkelt komponent tar inndata-argumenter, produserer utdata-filer og utfører en fase av pipelinen.
- **Bestemme sekvens**
  Her spesifiseres sekvesen pipelinen skal kjøre i. Dette defineres gjennom et Python *domain-specific language (DSL)*.[15] Et steg i pipelinens definisjon vekker en komponent i pipelinen. Om den er svært kompleks kan komponenter også kjøre iterativt, eller kun kjøres etter planlagte hendelser.
- **Inndata-parametere**
  En pipeline behøver et sett av inndata-parametere. Disse verdiene sendes til alle pipelinens komponenter, og innehar blant annet kriterier for hvordan data skal filtreres, og hvor man skal lagre artefaktene som pipelinen produserer.

Om man velger å kjøre Kubeflow Pipelines integrert med Google Cloud

Platform kan det se ut som i figur 4. Dette prosjektets løsning er ikke like tett integrert med tjenestene i GCP, men det er en fordel å ha oversikt over mulig integrasjon om bedriften ønsker en større sammenkobling i fremtiden. Flytskjema er inspirert av figur i Google Cloud Platforms dokumentasjon.[5]



Figur 4: Kubeflow Pipelines integrert med Google Cloud Platforms ML-verktøy

4.2 Verktøy og ressurser

Som det tidligere har blitt nevnt så utvikles løsningen i Google Cloud Platform, og oppdragsgiver har valgt å dele dette inn i tre forskjellige GCP-prosjekt, videre omtalt som *miljø*. Disse består av utvikling-, akseptansetest- og produksjonsmiljø. Dette avsnittet vil først beskrive hva et prosjekt i Google Cloud Platform er, før det går inn på de tre miljøene denne utviklingsoppgaven er delt inn i. Implementasjon og drift av løsningen er som beskrevet i forstudierapport[25] ikke inkludert i oppgavens omfang, og vil derfor ikke tas hensyn til.

4.2.1 Prosjekter i Google Cloud Platform

Et prosjekt i Google Cloud Platform er en måte å organisere bedriftens ressurser på. Det består av et gitt antall brukere, API-er med innstillinger for autentisering og monitoring, samt en oversikt over hvor mye penger

prosjektet har brukt. Brukere har altså valget mellom å sette alle sine ressurser i et enkelt prosjekt, eller å fordele og sortere dem utover flere. Hos en stor organisasjon som Statens vegvesen har sistnevnte vært den beste metoden. Dette forenkler prosessen med å sikre at brukere kun har tilgang til de prosjektene de er involverte i, slik at man ikke ødelegger viktige eller urelaterte bedriftsressurser om man gjør en feil innad eget prosjekt. GCP har noe som kalles Identity and Access Management (IAM). Dette er prosjektes og organisasjonens «tilgangskontroll». Tilgang kan gis i forskjellige grader til ansattes Google-kontoer, eller til såkalte Service Accounts. Disse kontoene tillater applikasjoner å autentisere seg på tvers over Google Cloud-ressurser og tjenester.[28] Eksempelvis om man ønsker å flytte data fra en applikasjon til en 'bucket', så kan en Service Account være nyttig. I dette prosjektet er det som nevnt innledningsvis tildelt tre prosjekter, disse følger fremgangsmåten bak 'Utvikling, testing, akseptanse og produksjon (DTAP)[29]', som består av følgende steg:

- **Utvikling**
  Løsningen utvikles i et uviklingssystem, dette behøver ikke å være innad prosjektet. Dette systemet behøver ikke å ha noen testmuligheter.
- **Testing**
  Når utvikleren mener at løsningen er klar, leveres den til testmiljøet for å verifisere at den fungerer som forventet. I denne løsningen har det vært nødvendig å teste såpass iterativt, at utviklings- og testmiljø har gått noe om hverandre.
- **Akseptanse**
  Når testingen er suksessfull leveres løsningen til et akseptansemiljø. Her vil bedriften sikre at løsningen møter deres krav.
- **Produksjon**
  Når bedriften er fornøyd med løsningen, kan den implementeres eller tas i bruk. Dette skjer i produksjonsmiljøet.

4.2.2 Utviklingsmiljø

Prosjektgruppen har fått tildelt tre miljøer i Google Cloud Platform fra Statens vegvesen. Det første fungerer som et utviklingsmiljø, der forslag til løsning blir skapt. Selv om et utviklingsmiljø i teorien ikke behøver testmuligheter, så har dette vært et behov i denne sammenhengen. Dette har skjedd på bakgrunn av at flere verktøy for utvikling av pipeliner ligger inne i GCP, og det blir svært vanskelig å komme frem til en løsning kun basert på lokale utviklingsplattformer.

Jupyter Notebook

Det mest brukte lokale verktøyet for utvikling av løsningen har vært Jupyter Notebooks. Dette verktøyet kan kan kjøres gjennom AI Hub og virtuelle maskiner (instanser) på GCP, eller på lokal datamaskin. Jupyter Notebooks er en nettbasert applikasjon med åpen kildekode, som lar brukeren opprette og dele dokumenter som inneholder blant annet kode, likninger og virtualiseringer. Dette egner seg godt til å utvikle og trene maskinlæringsmodeller lokalt, og brukes derfor mye av datavitere.[27] Teknologien er godt integrert med Google Cloud Platform, noe som gjør det lett for datavitere å enten bruke notisbøker rett i GCP, eller å overføre ferdige notisbøker til egen AI Hub når de skulle ønske. I dette prosjektet har Jupyter Notebooks blitt brukt til å blant annet skrive og kompilere pipelinene.

4.2.3 Akseptanse- og testmiljø

Når løsningen er ferdig utviklet skal den inn i akseptanse- og testmiljø. Disse miljøene er vanligvis separerte, men i denne sammenhengen er de slått sammen. Her skal man resprodusere løsningen, altså pipelinen, med minst mulig komponenter. Målet er å se om man kan reprodusere løsningen enkelt, og bruke flere modeller og/eller pipeliner i samme prosjekt. Løsningen er altså klar, men man må oppdage hvordan den best mulig kan implementeres.

4.2.4 Produksjonsmiljø

Da oppgaven er avgrenset til utvikling og testing, men ikke implementasjon, utgår dette miljøet i denne sammenhengen. Ved videre arbeid med løsningen vil det involveres, men dette gjennomgås ikke i denne rapporten.

4.3 Detaljerte løsningsbeskrivelser

For å tilby en kontekst for løsningsbeskrivelser vil en overordnet liste over deloppgaver som må løses presenteres, som videre peker på tekniske løsninger på deloppgavene. Videre presenteres de tekniske løsningene som omfattes i detalj.

1. **Nye versjoner av modell**
   Det er ønskelig å kunne deploye nye versjoner av maskinlæringsmodeller
   Løsning: TFX Evaluator, TFX Pusher
2. **Eksponere maskinlæringsmodell**
   Maskinlæringsmodellen må kunne eksponeres for input og brukes fra andre applikasjoner
   Løsning: TFX Pusher, AI Platform: Models
3. **Overvåke maskinlæringsmodell**
   Det er ønskelig å kunne overvåke maskinlæringsmodellen og dens evne til å predikere riktig
   Løsning: GCP logging og monitoring
4. **Skalere pipeline**
   Pipeline må kunne skaleres med tanke på kapasitet og ytelse
   Løsning: Kubernetes Engine i GCP
5. **Trene maskinlæringsmodeller effektivt**
   Maskinlæringsmodeller må kunne trenes i GCP
   Løsning: TFX Trainer, AI Platform: Jobs
6. **Lagre maskinlæringsmodeller**
   Maskinlæringsmodeller må kunne lagres på en hensiktsmessig måte
   Løsning: TFX Pusher, AI Platform: Models

4.3.1 TensorFlow Extended (TFX)

Her kommer detaljert beskrivelse av de TFX-komponenter som er direkte relevant for deloppgavene i kapittel 4.3. En gjennomgang av hvordan TFX-komponentene henger sammen, samt en forklaring av de øvrige komponentene er å finne i kapittel 4.3.4.

4.3.1.1 TFX Evaluator Evaluator analyserer modellen som har blitt trent i pipelinen og ser hvordan den yter på evalueringssettet. Denne kan man konfigurere for å se nærmere på ytelse i spesifikke tilfeller. I forstudiet så vi på muligheten for å velge den modellen som yter best i det spesifikke tilfellet den skal brukes til, og beslutningsgrunnlaget for dette gis fra TFX Evaluator. Evaluator kan også brukes til å sammenligne den nye modellen med den gamle modellen, og automatisk velge å deploye den nye modellen eller skrote den.

4.3.1.2 TFX Pusher TFX Pusher deployer modellen fra pipelinen til et endepunkt. I dette tilfellet deployes modellen til AI Platform: Models. Modellen som deployes er den som blir godkjent av TFX Evaluator.

4.3.1.3 TFX Trainer Trainer utfører selve treningen av modellen. Komponenten tar inn data og konfigurasjon, og trener en modell ut i fra dette. Denne kan konfigureres til å kjøre treningen i AI Platform: Jobs, og er ønskelig å gjennomføre i dette prosjektet. Konfigurasjon av treningen må gjøres av en dataviter, så bachelorgruppen må lage en løsning for hvordan dataviteren skal få lagt inn konfigurasjonen.

4.3.2 Google Cloud Platform (GCP)

4.3.2.1 AI Platform AI Platform i GCP tilbyr en rekke funksjonalitet for maskinlæring på skyen. På AI Platform finner man alt man trenger fra å skape modeller, trene dem og eksponere dem. Funksjonalitet herfra skal integreres i pipelinen for å integrere funksjonalitet med resten av GCP.

Models Her lagres trente maskinlæringsmodeller. Disse kan man eksponere for tredjepartsapplikasjoner med API-kall som er ferdig laget i GCP. Dette gjør at man slipper å lage API-kallene selv. Pipelinen skal deploye modeller hit på slutten av pipelinen. Versjoner grupperes etter modell, og man kan spesifisere en standardversjon for hver modell.

Pipelines Under pipelines ligger instanser av Kubeflow Pipelines. Hver instans kan ha flere pipeliner og modeller, men det kan være hensiktsmessig

å skille disse i forskjellige instanser. Mer om dette i kapittel 4.3.3.

Jobs Her skjer selve treningen av modeller i AI Platform. Pipelinen skal kjøre treningen her fremfor å gjøre det lokalt på orkestratoren. All konfigurasjon av treningen vil komme fra TFX Trainer.

Notebooks Her ligger Jupyter Notebooks. Dette skal brukes som utviklingsmiljø for å utvikle pipeliner og deploye dem til orkestratoren. Når man oppretter en notebook, har den allerede alt man trenger av verktøy for å drive med maskinlæring installert, og gir på den måten et miljø som er klart til bruk og enkelt å reprodusere.

4.3.2.2 Logging og monitorering

En fordel med å integrere GCP i så mange ledd som mulig i pipelinen, er at logger samles på ett sted. Logging i GCP samles på ett sted og har et eget spørrespråk for å finne frem til logger. Ved hjelp av dette kan logger fra de forskjellige stegene i pipelinen leses av.

4.3.2.3 Kubernetes Engine

Her tilbys et kontainerbasert miljø for deploying av applikasjoner. I dette prosjektet er selve pipelinen som kjører i Kubeflow Pipelines å anse som en applikasjon som kjører i Kubernetes Engine. Applikasjoner nytter clustere for porsjonering av kapasitet.

4.3.3 Kubeflow Pipelines (KFP)

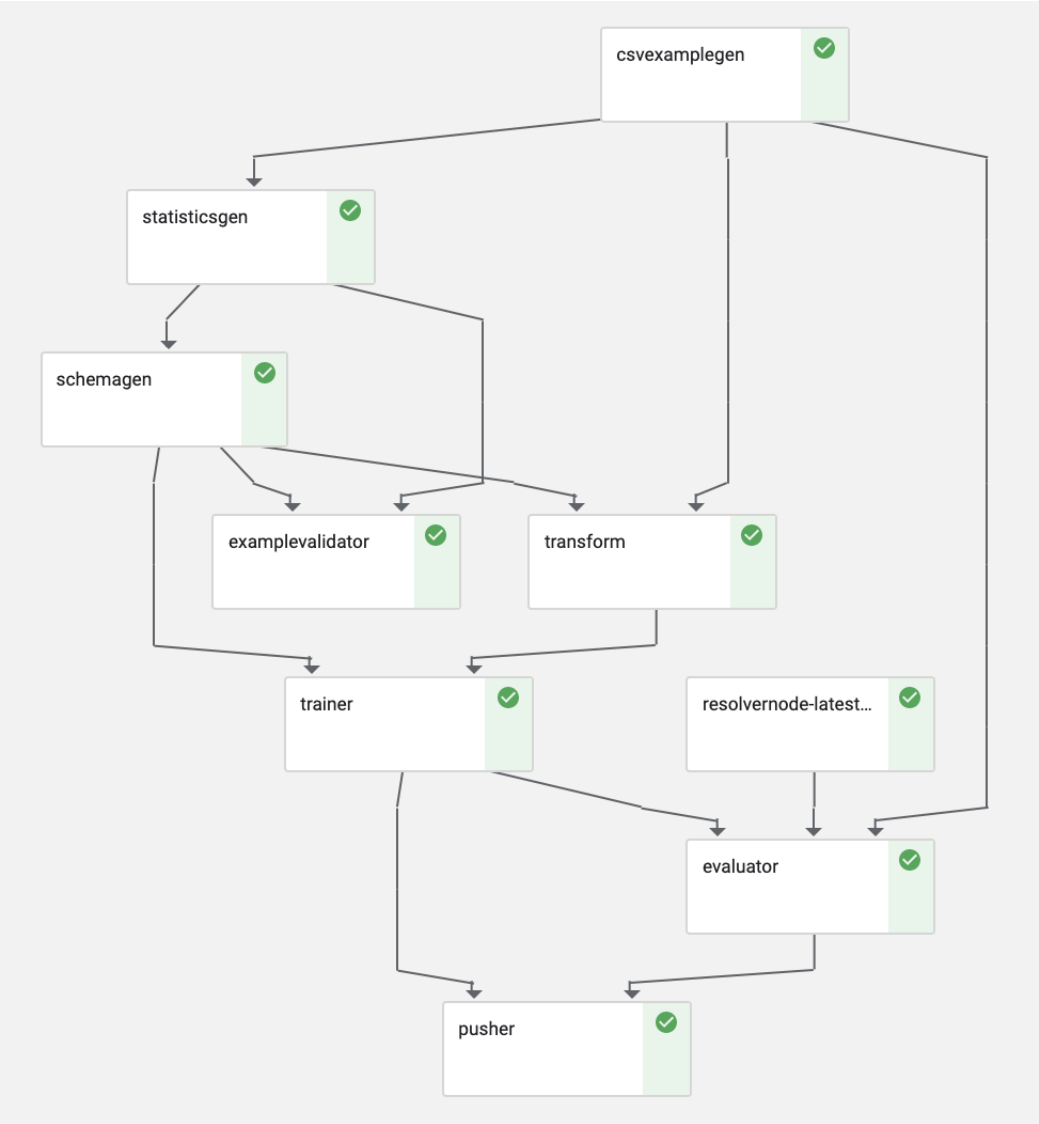
Rent teknisk konfigureres pipelinen i TFX, hvor den blir satt opp til å bruke komponenter og funksjoner i GCP. Videre kompileres TFX-koden til en konfigurasjonsfil som er lesbar for KFP. Deretter opprettes pipelinen i KFP, som kjører på et cluster i Kubernetes Engine på GCP. Dette betyr at GCP står for alt av nødvendig datakraft og at alt kan logges og monitoreres i GCP. Dette er svært nyttig da SVV allerede kjenner skyplattformen og benytter den i stor grad.

Valget om KFP som orkestrator er i hovedsak begrunnet i at det er god integrasjon mot GCP, men også at KFP er kompatibelt med andre ML-rammeverk enn TensorFlow. Dette legger til rette for å utvide støtte for

andre maskinlæringsmodeller i fremtiden. Øvrige argumenter omhandler skalerbarhet, noe Kubernetes Engine har god støtte for.

4.3.4 Flyt i pipelinen

Dette kapittelet skal beskrive flyten i pipelinen slik den blir konfigurert med TFX-komponentene beskrevet i kapittel 4.3.1.



Figur 5: TFX-komponentene satt sammen i Kubeflow

Examplegen er første stopp i pipelinen og er ansvarlig for inndataen. Dataen blir delt opp i trenings- og evalueringssett som blir brukt av Transform

og StatisticsGen. ExampleGen kan konfigureres til å hente data fra csv-filer eller direkte fra BigQuery.

Resolvernode er den andre inngangsnoden i pipelinen. Denne henter inn gjeldende versjon av modellen for å sammenligne den nye i Evaluator.

StatisticsGen genererer statistikk fra dataen fra examplegen.

Schemagen lager skjemaer av dataen fra examplegen. Skjemaet beskriver inndataen med typer, kategorier og rammer. TFX-komponenter for datavalidering, transformering og modellanalyse bruker skjemaet som kommer herfra. Skjemaet er automatisk generert og stemmer ikke nødvendigvis 100%, så dataviter bør se over og eventuelt modifiesere. Skjemaet kan skrives helt manuelt, men det er gjerne greiest å modifisere et som kommer fra denne komponenten. Skjemaet skrives i klartekst, noe som gjør det enkelt å redigere av datavitere.

Examplevalidator validerer inndataen fra examplegen mot skjemaet i schemagen og ser etter avvik. Resultatet fra denne komponenten brukes ikke videre i pipelinen.

Transform tar imot skjemaet fra Schemagen og data fra Examplegen og førprosesserer dataen ut ifra en funksjon som må forsynes av dataviter.[30] Denne funksjonen er spesifikk for hver modell og krever teknisk kompetanse innen maskinlæring, så prosjektgruppen skal ikke utvikle disse funksjonene, men legge til rette for at dataviteren kan legge inn sin.

Trainer utfører treningen, og tar inn utdataen fra Transform og SchemaGen, såvel som en konfigurasjonsfil for treningen som forsynes av dataviter. Etter Trainer har kjørt, kommer det ut en ferdigtrent modell.

Evaluator sammenligner den nylig trente modellen med en baseline eller en tidligere versjon for å vurdere ytelsen mot hverandre. Denne kan konfigureres

til å se på mindre segmenter av dataen for å sammenligne ytelsen i spesielle tilfeller. Evaluator returnerer en vurdering av beste modell.

Pusher tar modellen fra Trainer og vurderingen til Evaluator og deployer modellen til endepunktet. Dette er siste stopp i pipelinen.

4.3.5 Forutsetninger og avhengigheter

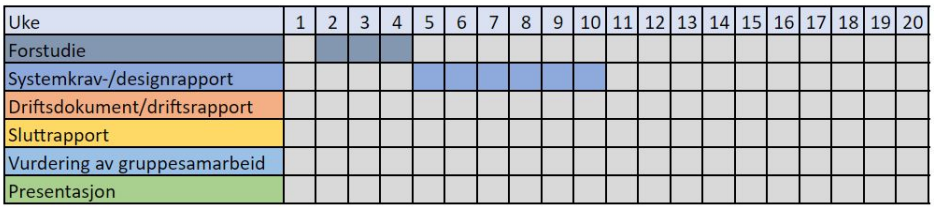
- Google Cloud Platform
- Brukerkonto eller Service Account tilknyttet prosjekt i GCP
- Maskinlæringsmodell skrevet i TensorFlow
- Konfigurerte moduler for TFX (trainer og transform)

4.3.6 Programvare og systemkrav

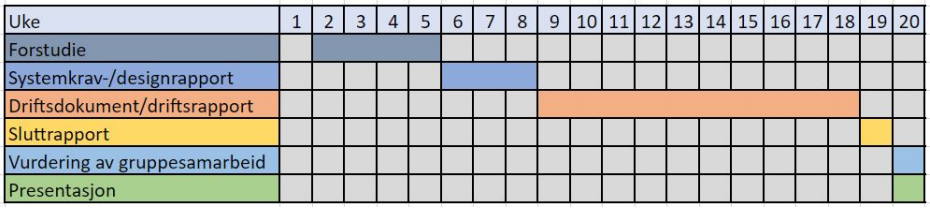
- TensorFlow >= 2.3.2
- TFX >= 0.26.1
- KFP >= 1.4.0
- Google Cloud SDK >= 327.0.0
- Python >= 3.7 < 3.9
- Kommandolinjegrensensnitt; CMD (Win), Bash (Lin), Bash/Terminal (Mac)

4.3.7 Oppdatert GANTT-diagram

Oppdatert GANTT-diagram ligger på prosjektsiden i Teams. Om leser ikke har tilgang til denne kan kopi forespørres. Figur 6 representerer den faktiske arbeidsprosessen, og figur 7 viser det som var planlagt fra start. Som man ser har designrapporten vært mer omfattende enn forventet, noe som er naturlig i et slikt innovativt prosjekt.



Figur 6: Faktisk arbeidsprosess.

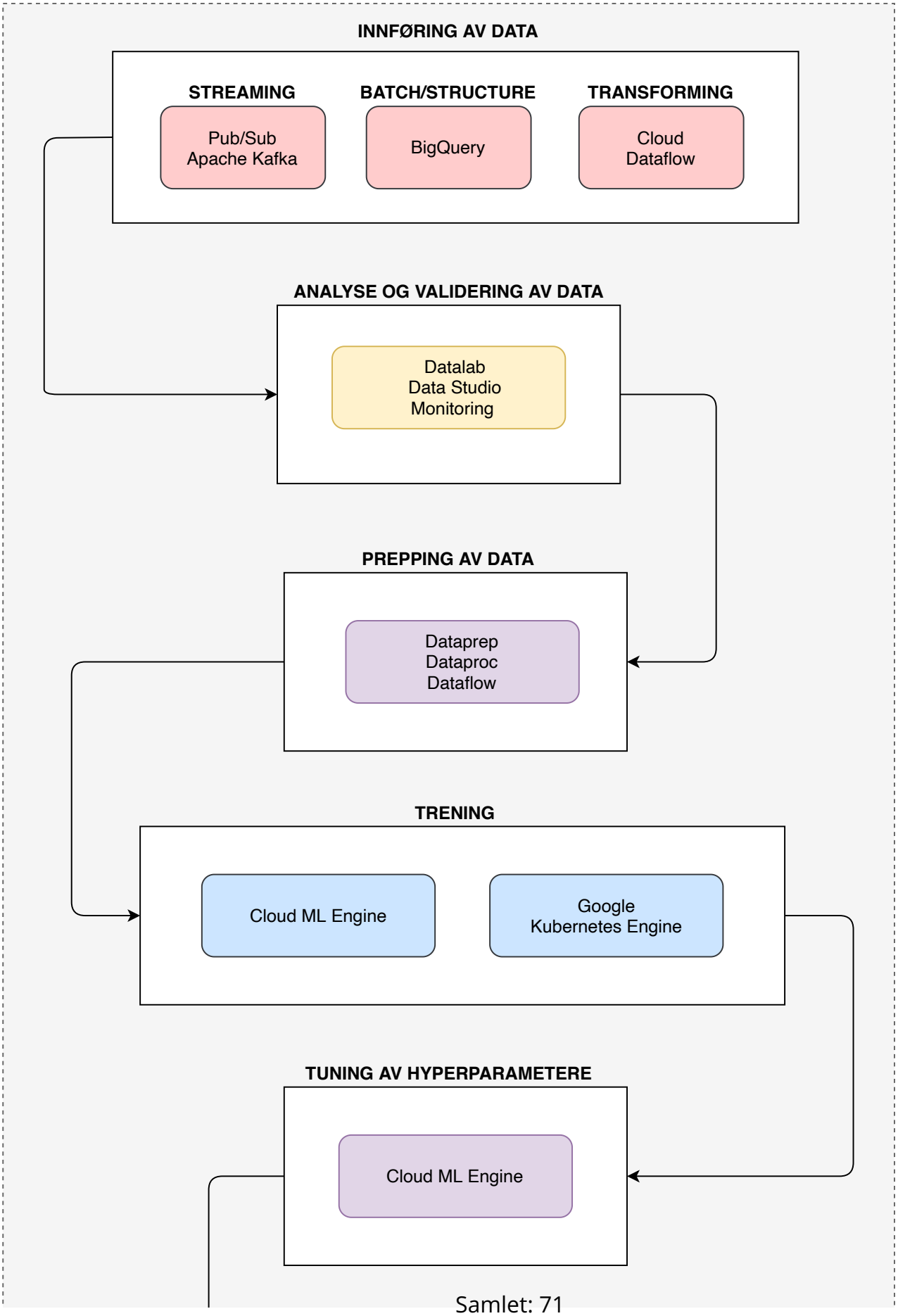


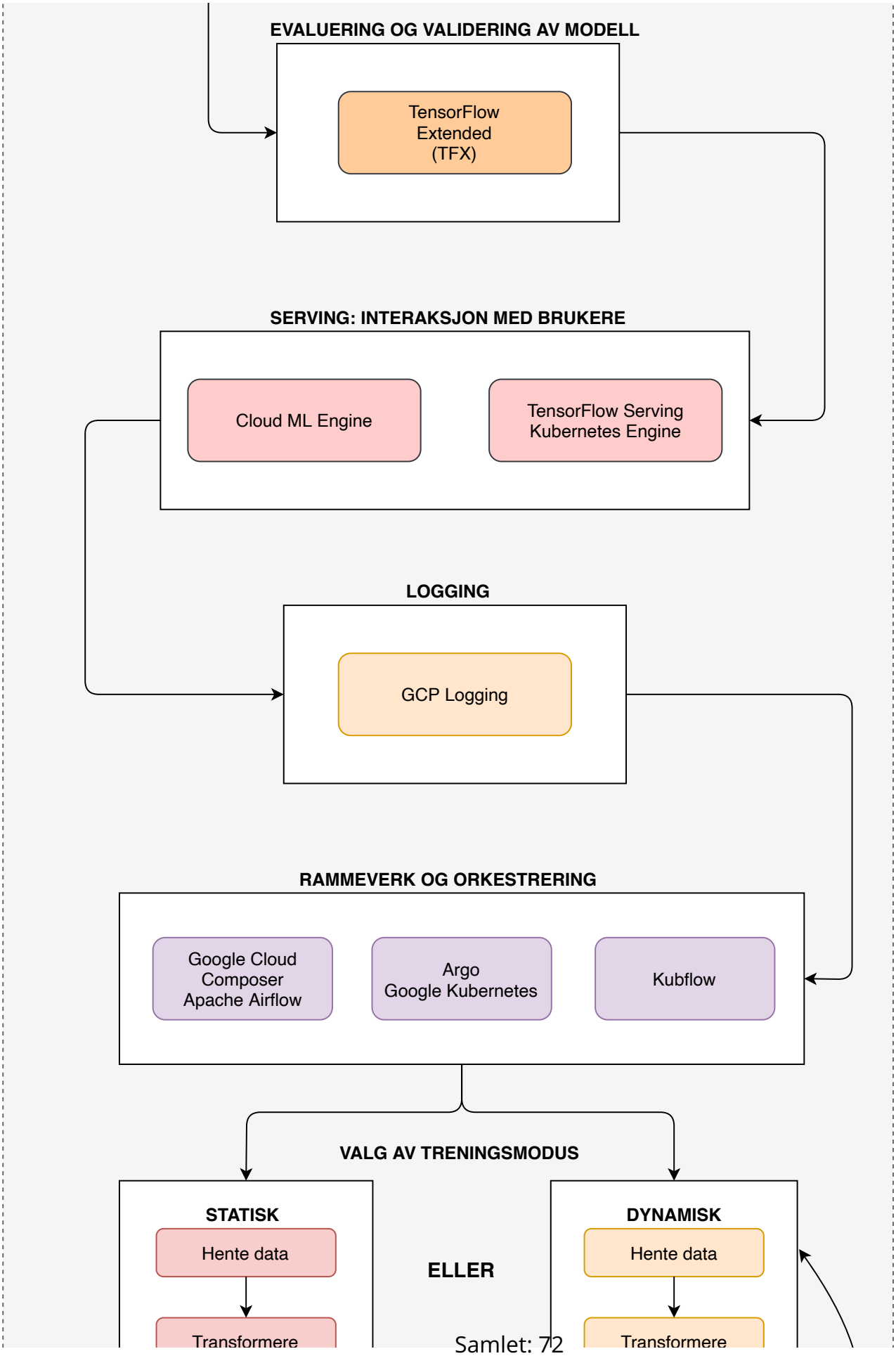
Figur 7: Planlagt arbeidsprosess.

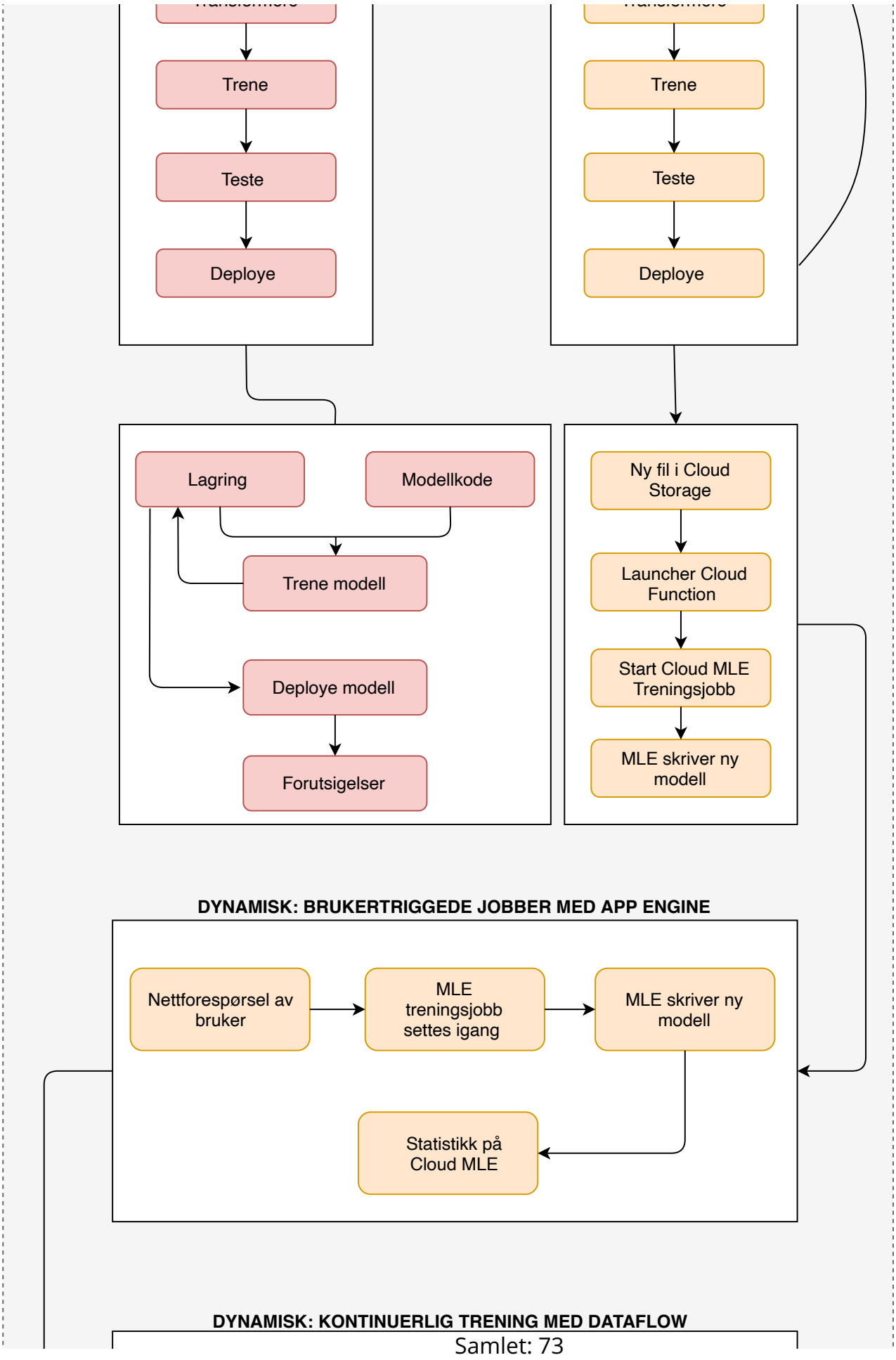
5 Alternative flyter i GCP

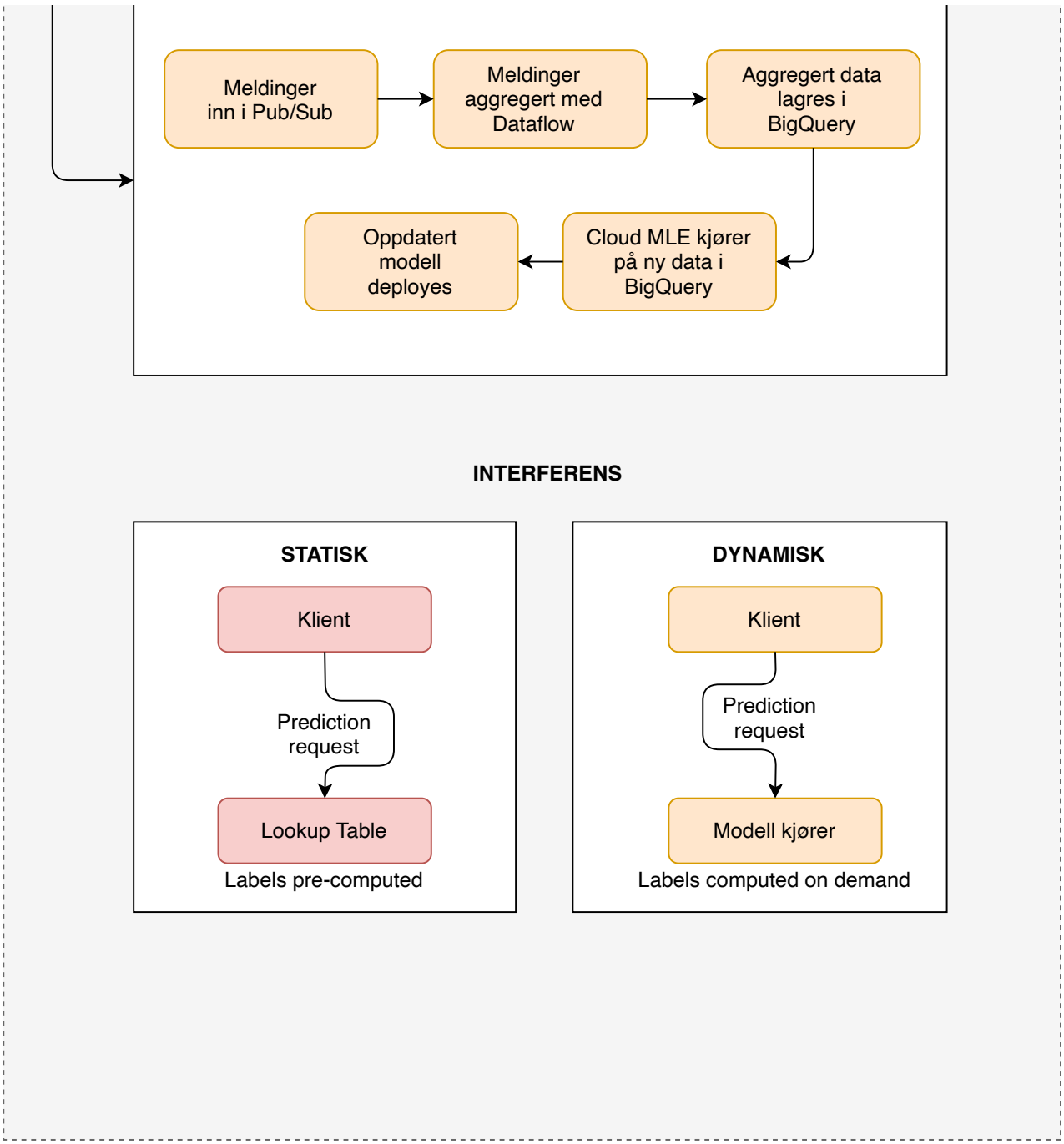
Ut over valgt teknologi og løsning har prosjektgruppen utarbeidet en oversikt over alternative flyter for pipeline i Google Cloud Platform som kan være til nytte for vurdering av andre løsninger. Dette er kun en overordnet oversikt og vil ikke beskrives ytterligere i dybden.

ML-ARKITEKTUR




EVALUERING OG VALIDERING AV MODELL
TensorFlow Extended (TFX)
SERVING: INTERAKSJON MED BRUKERE
Cloud ML Engine
TensorFlow Serving Kubernetes Engine
LOGGING
GCP Logging
RAMMEVERK OG ORKESTRERING
Google Cloud Composer Apache Airflow
Argo Google Kubernetes
Kubflow
VALG AV TRENINGSMODUS
STATISK
Hente data
Transformere
ELLER
DYNAMISK
Hente data
Transformere


Trene
Teste
Deploye
Trene
Teste
Deploye
Lagring
Modellkode
Trene modell
Deploye modell
Forutsigelser
Ny fil i Cloud Storage
Launcher Cloud Function
Start Cloud MLE Treningsjobb
MLE skriver ny modell
DYNAMISK: BRUKERTRIGGEDE JOBBER MED APP ENGINE
Nettforespørsel av bruker
MLE treningsjobb settes igang
MLE skriver ny modell
Statistikk på Cloud MLE
DYNAMISK: KONTINUERLIG TRENING MED DATAFLOW


Meldinger inn i Pub/Sub
Meldinger aggregert med Dataflow
Aggregert data lagres i BigQuery
Cloud MLE kjører på ny data i BigQuery
Oppdatert modell deployes
INTERFERENS
STATISK
Klient
Prediction request
Lookup Table
Labels pre-computed
DYNAMISK
Klient
Prediction request
Modell kjører
Labels computed on demand

Referanser

[1] L. Meisingseth *Presentasjon om Saga*, 2019.
Hentet fra:
https://www.vegvesen.no/_attachment/2849432/binary/1349698?fast_title=Saga-+Stordataplatform+for+transportdata.pdf.
Lastet ned: 15.02.2021

[2] Google Cloud, *MLOps: Continuous delivery and automation pipelines in machine learning*, 2020.
Hentet fra:
https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.
Lastet ned: 22.02.2021

[3] TensorFlow, *The TFX User Guide*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/guide.
Lastet ned: 23.02.2021

[4] V. Tatan, "Intro to ML Ops: Tensorflow Exntended (TFX)", *towards data science*, Apr. 2020. [Online].
Hentet fra:
https://towardsdatascience.com/intro-to-ml-ops-tensorflow-extended-tfx-39b6ab1c7
Lastet ned: 23.02.2021

[5] Google Cloud, *Architecture for MLOps using TFX, Kubeflow Pipelines, and Cloud Build*, 2021.
Hentet fra:
https://cloud.google.com/solutions/machine-learning/architecture-for-mlops-using-tfx-kubeflow-pipelines-and-cloud-build.
Lastet ned: 23.02.2021

[6] GitHub: TensorFlow, *TensorFlow Data Validation*, 2021.
Hentet fra:
https://github.com/tensorflow/data-validation.
Lastet ned: 23.02.2021

[7] TensorFlow, *Get Started with Tensorflow Transform*, 2021.
Hentet fra:
`https://www.tensorflow.org/tfx/transform/get_started`.
Lastet ned: 23.02.2021

[8] TensorFlow, *Estimators*, 2021.
Hentet fra:
`https://www.tensorflow.org/guide/estimator`.
Lastet ned: 23.02.2021

[9] TensorFlow, *Getting Started with TensorFlow Model Analysis*, 2021.
Hentet fra:
`https://www.tensorflow.org/tfx/model_analysis/get_started`.
Lastet ned: 23.02.2021

[10] TensorFlow, *Serving Models*, 2021.
Hentet fra:
`https://www.tensorflow.org/tfx/guide/serving`.
Lastet ned: 23.02.2021

[11] Kubeflow, *Overview of Kubeflow Pipelines*, 2020.
Hentet fra:
`https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/`.
Lastet ned: 27.02.2021

[12] Argo Project, GitHub, *Argo Documentation*, 2021.
Hentet fra:
`https://argoproj.github.io/argo-workflows/`.
Lastet ned: 01.03.2021

[13] Kubeflow, *Building Pipelines with the SDK*, 2020.
Hentet fra:
`https://www.kubeflow.org/docs/pipelines/sdk/`.
Lastet ned: 01.03.2021

[14] Docker Docs, *docker image*, 2021.
Hentet fra:

https://docs.docker.com/engine/reference/commandline/image/.
Lastet ned: 01.03.2021

[15] Wikipedia, *Domain-specific language*, 2021.
Hentet fra:
https://en.wikipedia.org/wiki/Domain-specific_language.
Lastet ned: 01.03.2021

[16] Google Cloud, *Google Cloud Overview*, 2020.
Hentet fra:
https://cloud.google.com/docs/overview.
Lastet ned: 05.03.2021

[17] Google Cloud, *AutoML*, 2020.
Hentet fra:
https://cloud.google.com/automl/docs.
Lastet ned: 05.03.2021

[18] Google Cloud, *AI Platform*, 2020.
Hentet fra:
https://cloud.google.com/ai-platform.
Lastet ned: 05.03.2021

[19] Google Cloud, *AI Hub documentation*, 2020.
Hentet fra:
https://cloud.google.com/ai-hub/docs.
Lastet ned: 05.03.2021

[20] crossML engineering, "Google Cloud Platform (GCP) for Machine Learning AI", *Medium*, Jul. 2020. [Online].
Hentet fra:
https://medium.com/crossml/google-cloud-platform-gcp-for-machine-learning-ai-361
Lastet ned: 05.03.2021

[21] Google Cloud, *What is BigQuery ML?*, 2020.
Hentet fra:
https://cloud.google.com/bigquery-ml/docs/introductions.
Lastet ned: 05.03.2021

[22] Google Cloud, *Working with Cloud Storage*, 2020.
Hentet fra:
https://cloud.google.com/ai-platform/training/docs/working-with-cloud-storage.
Lastet ned: 05.03.2021

[23] Google Cloud, *Dataflow*, 2020.
Hentet fra:
https://cloud.google.com/dataflow.
Lastet ned: 05.03.2021

[24] Google Cloud, *Dataproc*, 2020.
Hentet fra:
https://cloud.google.com/dataproc.
Lastet ned: 05.03.2021

[25] J. A. Langholm, I. Andersen og E. F. Dalen, "Forstudierapport", *NTNU*, Feb. 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[26] J. A. Langholm, I. Andersen og E. F. Dalen, "Ordbok", *NTNU*, Feb. 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[27] Jupyter, *The Jupyter Notebook*, 2020.
Hentet fra:
https://jupyter.org/.
Lastet ned: 05.03.2021

[28] Google Cloud, *Projects*, 2021.
Hentet fra:
https://cloud.google.com/storage/docs/projects.
Lastet ned: 05.03.2021

[29] Wikipedia, *Development, testing, acceptance and production*, 2009.
Hentet fra:
https://en.wikipedia.org/wiki/Development,_testing,_acceptance_

and_production.
Lastet ned: 05.03.2021

[30] Microsoft, *Feature engineering in machine learning*, 2020.
Hentet fra:
https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/create-features

[31] towards data science, *Which deep learning framework is the best?*, 2020.
Hentet fra:
https://towardsdatascience.com/which-deep-learning-framework-is-the-best-eb51431

[32] J. A. Langholm, I. Andersen og E. F. Dalen, "Ordbok", *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[33] Kubeflow, *Kubeflow Katib: Scalable, Portable and Cloud Native System for AutoML*, 2021
Hentet fra:
https://blog.kubeflow.org/katib/
Lastet ned: 07.04.2021

[34] J. A. Langholm, I. Andersen og E. F. Dalen, "Driftsrapport", *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[35] J. A. Langholm, I. Andersen og E. F. Dalen, "Sluttrapport", *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

3 Driftsrapport

MLOps på Google Cloud Platform
Driftsrapport

Ingvild Andersen, Jon Akselberg Langholm, Erik Flæsen Dalen

19. mai 2021

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
12.05.2021	1.0	Førsteutkast	Erik F.D., Ingvild A. og Jon A. L.
13.05.2021	1.1	Mindre revideringer etter tilbakemelding fra veileder	Erik F.D., Ingvild A. og Jon A. L.
19.05.2021	1.2	Språkvask	Erik F.D., Ingvild A. og Jon A. L.

Innhold

1 Innledning **5**
1.1 Dokumentets hensikt 5
1.2 Avgrensning 5
1.3 Oversikt over dokumentet 6
1.4 Forkortelser og definisjoner 7

2 Kort om teknologi **7**
2.1 Overordede stadier 8
2.2 Plattform: Google Cloud Platform (GCP) 8
2.2.1 Prosjekter og tilgangskontroll 8
2.2.2 Storage Buckets 9
2.2.3 Virtual Private Cloud (VPC) network 9
2.2.4 Google Cloud API 9
2.3 Rammeverk: TensorFlow Extended (TFX) 10
2.4 Kubernetes 10
2.5 Orkestrering: Kubeflow Pipelines (KFP) 11

3 Oversikt over løsning **11**

4 Hva skal implementeres **14**

5 Implementasjonsfaser **14**
5.1 Oppsett av testmiljø 14
5.2 Utvikling 15
5.3 Implementering i bedriften 15
5.3.1 Tilpasning av pipeline til modell 15
5.4 Videre arbeid 16

6 Konklusjon **18**

Referanser **19**

7 Appendiks **25**

A Gjennomgang av kode **25**
A.1 template/pipeline/gcp_infrastructure.py 25

A.2 template/pipeline/pipeline.py 31
A.3 template/pipeline/configs.py 36

Figurer

2 TFX: Eksempel på flyt 10
3 Kubeflow Pipelines integrert med Google Cloud Platforms ML-verktøy 11
4 TFX-komponentene satt sammen i Kubeflow 13

1 Innledning

1.1 Dokumentets hensikt

Dokumentet er skrevet i forbindelse med bacheloroppgaven 'MLOps på Google Cloud Platform' i faget IDRI3001, utarbeidet av Ingvild Andersen, Jon Akselberg Langholm og Erik Flæsen Dalen i samarbeid med NTNU og Statens vegvesen.

Bakgrunnen for oppgaven er at Statens vegvesen ønsker å utforske muligheten for å innføre en produksjonspipeline for maskinlæringsmodeller i Google Cloud Platform, samt hvilken løsning som ville vært mest effektiv for formålet. Løsningsforslag skal utarbeides som et såkalt 'Proof of Concept'. Driftrapporten gir leseren en detaljert innføring i den tekniske utførelsen av dette løsningsforslaget, samt problemer som kan oppstå underveis. Hensikten er at oppgavestiller skal kunne reprodusere og få detaljkunnskap om løsningen, samt muligheten til å drifte, implementere og utvikle den videre. Rapporten tar utgangspunkt i at leser allerede har et godt teknisk grunnlag, og har noe kjennskap til Google Cloud Platform.

Driftsrapporten tar utgangspunkt i designrapporten.[29] Løsningen er ikke ferdig testet før denne fasen av prosjektet, og det tas derfor forbehold om løsningsendringer underveis. Dette skjer i samarbeid med bedrift og veileder.

1.2 Avgrensning

Dette avsnittet beskriver oppgavens avgrensning, nærmere bestemt hva løsningen skal og ikke skal omfatte. Statens vegvesen ønsker et «Proof of Concept» på en produksjonspipeline i Google Cloud Platform for maskinlæringsmodeller. Avgrensningen avklarer prosjektgruppens ansvar, samt produksjonspipelinens ønskede funksjonalitet.

Oppgaven omfatter:

- Hente inn data
- Validere data
- Transformere data

- Versjonskontrollere modeller
- Overvåke modeller
- Validere modeller
- Levere modeller og medfølgende prediksjoner

Oppgaven omfatter ikke:

- Utvikling av maskinlæringsmodeller
- Drift av løsningen
- Implementasjon av produksjonsklar løsning
- Funksjonalitet for å hente inn data i sanntid (kontinuerlig trening)
- Opprette bruker eller prosjekt i Google Cloud Platform

Det ble i forstudierapporten gitt forbehold om at ønsket funksjonalitet kunne utelukkes om det skulle vise seg at oppgaven ble for stor. Etter research under designstadiet i oppgaven, viste det seg at det skulle være mulig å implementere all ønsket funksjonalitet. En gjennomgang av hvordan bachelorgruppen har løst alle krav oppdragsgiver hadde, kan finnes i kapittel 6. En avgrensning, eller nærmere avklaring, som ble gjort under designstadiet var at pipelinen kun vil være kompatibel med Tensorflow-modeller, da rammeverket pipelinen er bygget på bruker Tensorflow-biblioteker eksklusivt.

1.3 Oversikt over dokumentet

Første del av dokumentet (1) beskriver dokumentets hensikt (1.1), avgrensning (1.2), oversikt (1.3) og forkortelser og definisjoner (1.4). Videre presenteres leser en gjennomgang av teknologien som løsningen tar i bruk (2). Helt konkret beskrives løsningens overordende stadier (2.1), skyplattformen den kjører i (2.2) og underliggende funksjoner, rammeverk (2.3), Kubernetes (2.4) og orkestrator (2.5). I neste del av rapporten får leseren presentert en overordnet beskrivelse av løsningen (3) og hva som skal implementeres (4). Femte kapittel tar så leser med inn i løsningens implementasjonsfaser (5), som består av oppsett av testmiljø (5.1), utvikling (5.2), implementering i bedriften (5.3) og forslag til videre arbeid for

oppgavestiller (5.4). Til slutt gis det en konklusjon på rapporten i sin helhet (6). Det er også lagt til en appendiks (7) som inneholder kodegjennomgang (A).

1.4 Forkortelser og definisjoner

Se egen ordbok.[30]

2 Kort om teknologi

Følgende avsnitt presenterer en kort gjennomgang av teknologien som løsningen består av. Dette er hentet ut fra designrapporten[29] som ble utarbeidet tidligere i prosjektet, slik at driftsrapporten kan fungere som et enkeltstående dokument for leser. Ønskes det en mer detaljert gjennomgang enn det som presenteres her, så er dette å finne i designrapporten.[29] Prosjektgruppen kom frem til denne teknologien ved å ta utgangspunkt i krav fra Statens vegvesen, samt hensyn til hva som måtte til for å best mulig implementere prinsippene bak MLOps.[29] Disse prinsippene legger opp til automatisering hele veien fra maskinlæringsmodellene utvikles, til de leverer prediksjoner fra et endepunkt. Google Cloud Platform lagrer og kjører løsningen og lagrer dens underliggende data. Denne plattformen innehar en stor mengde maskinlæringsverktøy.

En automatisert maskinlæringspipeline er blant annet avhengig av komponenter, et rammeverk og en orkestrator. Rammeverket konfigurerer pipelinen, og orkestratoren kontrollerer kjøringen av denne. Den styrer og dirigerer rekkefølgen på komponentene og arbeidet som pipelinen består av. I denne løsningen består rammeverket av TensorFlow Extended (TFX), da det kan tilpasses modeller basert på populære TensorFlow. TFX er et ende-til-ende verktøy, og støtter opp under alt fra førprosessering av data til trening og levering av produksjonsklare modeller. Rammeverket støtter også opp under prinsippene bak MLOps. Løsningens orkestrator kalles Kubeflow Pipelines, som er en plattform for å bygge og kjøre ut portable og skalerbare maskinlæringspipeliner basert på Docker-konteinere. Orkestratoren er en ende-til-ende løsning som blant annet legger til rette for gjenbruk av kode, pipeliner og komponenter.

2.1 Overordede stadier

Løsningen skal som nevnt så godt det lar seg gjøre innføre prinsippene i MLOps. Den er bygget opp med utgangspunkt i følgende stadier:

1. Hente ut data
2. Analysere data
3. Forberede data
4. Trening av modellen
5. Evaluering av modellen
6. Validering av modellen
7. Levering av modellen
8. Overvåking av modellen

Videre gjennomgang av teknologiløsninger vil så nært som mulig gjenspeile disse stadiene.

2.2 Plattform: Google Cloud Platform (GCP)

En offentlig skytjeneste som leveres av Google. Tilbyr et stort antall datatjenester, og har spesielt mye å tilby innen maskinlæringsverktøy. Statens vegvesen er allerede kjent med GCP, og ønsket derfor en løsning som er integrerbar med denne plattformen. Det er her man finner prosjektets utviklings-, akseptansetest-, og produksjonsmiljø. For å kunne gjenskape løsningen slik det gjennomgås i dette dokumentet, behøver leseren en bruker hos Google Cloud Platform. Denne brukeren må være godkjent for betaling da det forekommer kostnader ved å kjøre løsningen i GCP.

2.2.1 Prosjekter og tilgangskontroll

For å kunne sette opp denne løsningen behøves et prosjekt i Google Cloud Platform. Slike prosjekter brukes på plattformen for å organisere ressurser i hvert sitt enkeltstående miljø. Det består av et gitt antall brukere, API-er med innstillinger for autentisering og monitorering, samt en oversikt over

prosjektets nåværende og estimerte fremtidige kostnad. Som bruker har man valget om å sette alle sine ressurser i et enkelt prosjekt, eller fordele dem over flere. En større organisasjon bør sikte på sistnevnte, da man kan sikre at brukere kun har tilgang til de prosjektene de er involverte i, slik at de ikke kan ødelegge ikke-relaterte bedriftsressurser om man gjør en feil innad eget prosjekt.[27] Prosjektets tilgangskontroll kalles Identity and Access Management (IAM). Tilgang kan gis til brukere og deres Google-kontoer, eller til Service Accounts. Disse tillater applikasjoner å autentisere seg på tvers av Google Cloud Platform sine ressurser og tjenester.[21]

2.2.2 Storage Buckets

All data som lagres i Google Cloud Platform må ligge i en såkalt *storage bucket*. Dette er konteinere som inneholder brukerens data. Det er ingen begrensning på hvor mange buckets man kan opprette, og man kan sette individuell adgangsbegrensning på hver av dem.[22] I denne gjennomgangen brukes buckets i hovedsak til lagring av maskinlæringsmodellens trenings- og testdata, samt artefakter. Sistnevnte er metadata fra kjøring av TFX-komponentene.

2.2.3 Virtual Private Cloud (VPC) network

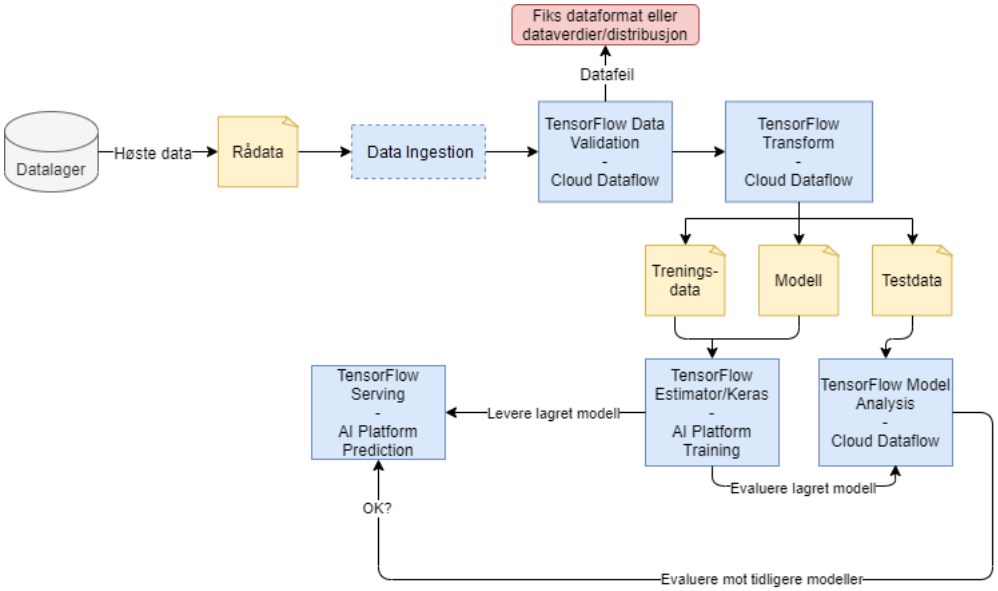
Et VPC-nettverk er et virtuelt privat nettverk som settes opp for at GCP-komponentene som inngår i løsningen skal kunne kommunisere med pipelinen, som ligger på et cluster i Kubernetes Engine. Et prosjekt kan inneha flere VCP-nettverk, og nye prosjekter starter med et standard nettverk som har et subnett i hver region.[23]

2.2.4 Google Cloud API

De ulike tjenestene i Google Cloud Platform har egne API-er som må aktiveres før de kan tas i bruk. GCP tilbyr også REST API-tjenester, som åpner for å kommunisere med GCP fra ønskede programmeringsspråk. Dette brukes blant annet i prosjektet for å opprette nødvendig infrastruktur gjennom et Pythonskript. Alle API-ene er av sikkerhetsmessige årsaker som standard deaktivert ved prosjektopprettelse.

2.3 Rammeverk: TensorFlow Extended (TFX)

TensorFlow Extended (TFX) tilbyr et rammeverk for å opprette pipeliner bestående av TFX-komponenter. Rammeverket støtter alt fra transformasjon av data til trening og levering av produksjonsklare modeller.[18] TFX har flere underliggende biblioteker som støtter opp under ønsket pipeline-funksjonalitet, slik som trening og utkjøring, samt integrasjon med GCP. Tjenestene som inngår i TensorFlow Extended kan legges nært opp mot de overordnede stadiene som er nevnt ovenfor (se figur 2).



Figur 2: TFX: Eksempel på flyt

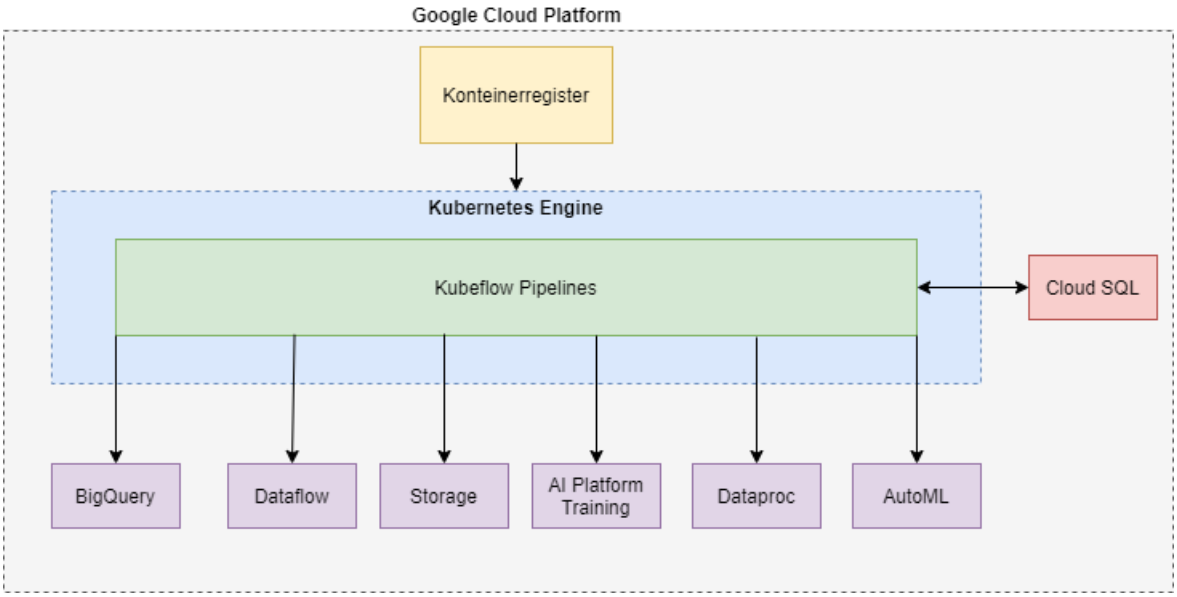
2.4 Kubernetes

Kubernetes er en plattform som kan kjøre ut, skalere og administrere applikasjoner som kjører i konteinere.[24] Videre kan man opprette noe som kalles Kubernetes cluster, som tillater slike konteinere å kjøre på tvers av virtuelle, fysiske og skybaserte miljø og maskiner. I denne løsningen kjøres det ut et cluster på Kubernetes Engine, som er en integrert tjeneste i Google Cloud Platform. Kubernetes Engine lar brukeren utnytte alle mulighetene ved Kubernetes direkte i GCP.[25] Løsningens orkestrator, som beskrives i neste avsnitt, kjører på et slikt cluster.

2.5 Orkestrering: Kubeflow Pipelines (KFP)

Orkestratoren behøves for å koble sammen de ulike komponentene i produksjonspipelinen. Dette er det som sørger for at komponentene kjøres automatisk, i ønsket sekvens, etter planlagte triggere. Kubeflow stammer fra Kubernetes, og har en underliggende tjeneste kalt Kubeflow Pipelines som spesielt er utviklet for maskinlæring. Hver komponent i KFP kan kjøre enten i Kubeflow eller i Google Cloud Platform. Argo Workflows[19] brukes for å orkestrere Kubernetes-ressursene, og KFP har en Python SDK[20] for å definere og manipulere komponentene. KFP kan om ønskelig kjøres tett integrert med maskinlæringstjenestene i GCP.

3 Oversikt over løsning



Figur 3: Kubeflow Pipelines integrert med Google Cloud Platforms ML-verktøy

Teknologiene som er beskrevet i kapittel 2 skal sys sammen for å samarbeide. De individuelle komponentene må også konfigureres, slik at disse til slutt utgjør et system som tilfredsstiller oppdragsgivers behov. Disse er:

- Kunne kjøre ut en maskinlæringsmodell i GCP.
- Kunne kjøre ut nye versjoner av en maskinlæringsmodell i GCP.

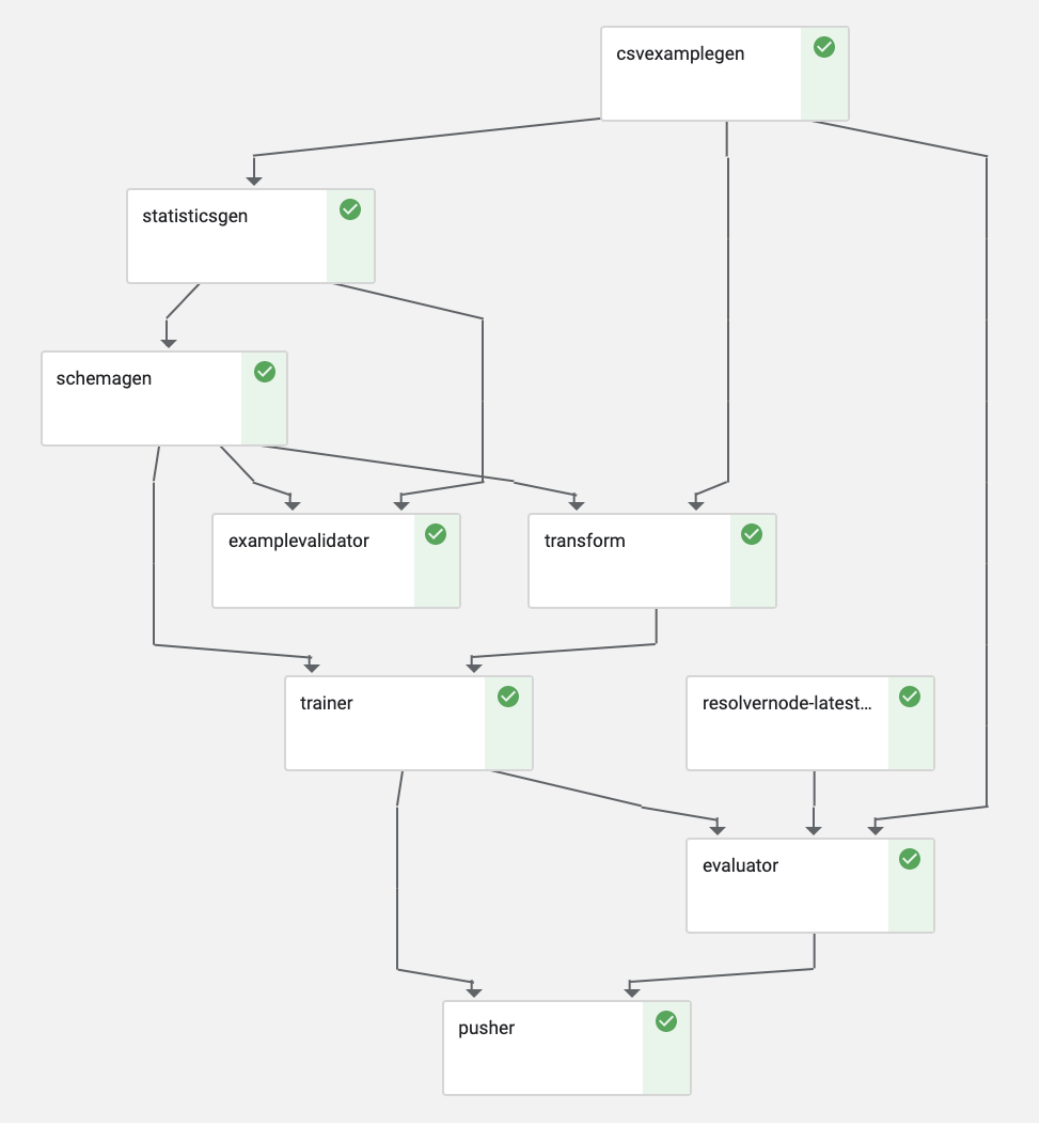
- Kunne eksponere en maskinlæringsmodell for input og bruke den fra andre applikasjoner.
- Finne ut hvordan man kan overvåke en maskinlæringsmodell og dens evne til å predikere riktig.
- Finne ut hvordan man kan skalere en pipeline med hensyn på kapasitet og ytelse.

Figur 3 viser hvordan alle komponentene henger sammen. På øverste nivå ligger GCP med innebygde funksjoner som blir brukt til forskjellige jobber i pipelinen. På GCP kjøres også Kubernetes engine. I Kubernetes engine ligger Kubeflow Pipelines som kjører pipelinen som er definert med Python-kode som bruker TFX-biblioteker. Orkestratoren Kubeflow Pipelines sørger for at de enkelte nodene i pipelinen kjører i riktig rekkefølge. Pilene som peker på de lilla boksene, illustrerer at de individuelle komponentene av pipelinen kjører sine jobber ved hjelp av forskjellige funksjoner som er innebygget i GCP.

Oppgaven som skal løses for å implementere dette systemet er todelt; en del av oppgaven er å sette opp arkitekturen rundt dette systemet som tillater de forskjellige komponentene å samarbeide. Den andre delen av oppgaven er å lage selve produksjonspipelinen som tar i bruk funksjonene i GCP.

Den første delen av oppgaven som går ut på å sette opp arkitekturen i GCP hvis løsning er beskrevet i kapittel A.1 og i vedlagt README.[31]

Del to av oppgaven som omfatter oppsett av selve produksjonspiplinen består av ren Python-kode. Denne koden definerer hvilke jobber som skal gjøres, hvordan disse skal gjøres, og rekkefølgen på dem. Figur 4 viser flyten i denne pipelinen.



Figur 4: TFX-komponentene satt sammen i Kubeflow

Csvexamplegen, eventuelt *bigqueryexamplegen*, tar inndata for pipelinen. *Statisticsgen* tar imot dataen og genererer statistikk. *Schemagen* lager skjemaer for dataen basert på statistikken. *Examplevalidator* sjekker at dataen stemmer overens med skjemaet. *Transform* utfører førprosessering på data før trening. *Trainer* trener en maskinlæringsmodell. *Resolvernode* henter gjeldende modell. *Evaluator* sammenligner ytelsen til den nye modellen med den gamle på data fra *examplegen*. *Pusher* laster eventuelt opp den nye modellen om den tilfredsstiller kriterier stilt av *evaluator*.

En grundigere gjennomgang av flyten i pipelinen, begrunnelse, og forklaring

av de individuelle komponentenes rolle kan finnes i designrapporten.[29]

4 Hva skal implementeres

I bacheloroppgaven skal det implementeres en rekke komponenter som til sammen utgjør en pipeline som tilfredsstiller all funksjonalitet som er beskrevet i kapittel 1.2. Statens vegvesen benytter allerede GCP, så funksjonaliteten skal kjøre der.

Komponenter som må implementeres eller være klare for at pipelinen skal kjøre er følgende:

- En bucket i Google Cloud Storage
- Et VPC-nettverk
- Et Kubernetes-cluster
- Kubeflow Pipelines deployet til det overnevnte Kubernetes-clusteret
- Nødvendige Google Cloud Platform API-er må være aktivert

Etter ønske, kan BigQuery benyttes, men dette er ikke påkrevd for å kjøre pipelinen. Mer om dette i kapittel 5.4. Etter at disse komponentene er klargjorte, kan pipelinens kildekode kjøres og modifiseres.

Kapittel 2.2 har forklart disse forskjellige komponentene nærmere.

5 Implementasjonsfaser

5.1 Oppsett av testmiljø

Å sette opp et testmiljø kan gjøres på to måter - ved bruk av Jupyter Notebooks eller på lokal datamaskin. Det er smak og behag som avgjør hvilken metode man velger, men det anbefales likevel å bruke Jupyter Notebooks, da oppsettet er mest testet der og systemet kommer med nødvendige preinstallerte programmer. Spesielt dersom lokal datamaskin kjører Windows anbefales Jupyter Notebooks, da prosjektgruppen har

arbeidet på de Unix-baserte operativsystemene macOS Big Sur og Linux Debian Testing (Bullseye) når de ikke har brukt Jupyter Notebooks.

For oppsett av testmiljø/PoC henvises det til repositoryets README, som ligger vedlagt.[31]

Skriptet for oppsett av infrastruktur er forklart i appendiks A.1.

5.2 Utvikling

Mesteparten av arbeidet i utføringsfasen ligger her. Etter at alle komponenter lå til rette, kunne arbeidet med å utvikle pipelinen i Jupyter-miljøet starte. Jupyter-miljøet har alle avhengigheter installert fra før av, noe som forenkler arbeidet. Det har i hovedsak blitt utviklet to Python-filer som definerer pipelinen. `pipeline.py` definerer sammenhengen mellom alle komponentene i pipelinen, og `configs.py` inneholder alt av konfigurasjon. I utgangspunktet skal det ikke være nødvendig å endre på noe i `pipeline.py` med mindre man vil heller vil bruke BigQuery fremfor CSV som datainput.

Koden som ble utviklet i denne fasen gjennomgås i appendiks A.2 og A.3.

5.3 Implementering i bedriften

At det ikke er mulig å kopiere hele GCP-miljøer, kompliserer arbeidet med å ta i bruk produktet til bachelorgruppen. Målet med dette dokumentet og med README-filen[31] til github-repositoryet[1] er at bachelorgruppens produkt skal være raskt og enkelt å ta i bruk hos Statens vegvesen. Den eneste forutsetningen er at man bruker Google Cloud Platform.

5.3.1 Tilpasning av pipeline til modell

Pipelinen som er utviklet av bachelorgruppen har blitt utviklet for å være generisk, slik at den potensielt kan tilpasses en hvilken som helst modell. Det er tre ting som må skreddersys til den modellen som skal utvikles i pipelinen. Videre endringer av pipelinen er mulig, men ikke strengt nødvendig. De tre tingene som må være spesifikke for modellen er:

- Transform

- Trainer
- Evaluator

5.3.1.1 Transform

Transform utfører førprosessering på inndataen før den blir brukt videre til trening og evaluering av modellen. Her må dataviter konfigurere slik at de attributter som blir brukt videre kommer med. Her utleder man gjerne også attributter som ikke allerede eksisterer som egne felter i inndataen. Det utvikles en modulfil som inneholder en funksjon som kalles av komponenten når komponenten kjører i pipelinen. Se A.3.0.3 for nøyere beskrivelse av hvordan denne skal konfigureres.

5.3.1.2 Trainer

Trainer er komponenten som utfører selve treningen. Denne må naturligvis også skreddersys til modellen den skal trene. På lik linje med Transform sin konfigurasjonsfil, skal Trainers konfigurasjonsfil også lagres i Google Cloud Storage og inneholde en funksjon som kalles av komponenten når pipelinen kjører. Nærmere instruksjoner er å finne i A.3.0.4.

5.3.1.3 Evaluator

Den siste komponenten som må tilpasses er Evaluator. Konfigurasjonen av Evaluator ligger som et objekt i `configs.py`. Her må man definere hvilke attributter de to modellene skal sammenlignes på, og hvor god en modell må være for å bli deployet. Nærmere forklaring ligger i A.3.0.6

5.4 Videre arbeid

Bacheloroppgaven har fullført et «Proof of Concept» og implementert den funksjonaliteten som Statens vegvesen ønsket seg. Pipelinen har allikevel forbedringspotensiale. Det er funksjonalitet som kan legges til pipelinen, men som ikke har blitt utviklet da gruppen ikke hadde tid til dette, og prioriterte heller å få på plass den funksjonaliteten som ble bedt om fra starten av.

Det første som må gjøres for å ta i bruk pipelinen, forutenom å sette opp infrastrukturen i GCP, er å tilpasse pipelinen til den konkrete modellen som

skal trenes. Kapittel 5.3.1, samt de relevante underkapitlene i A.3 beskriver hvordan dette gjøres.

Etter kommunikasjon med maskinlæringsteamet senere i prosjektet, har det blitt etterspurt muligheten for å kjøre pipelinen lokalt for å enklere kunne teste pipelinen i det den tilpasses til modellen. En `local_runner.py` som gjør dette, ble ferdigstilt av maskinlæringsteamet 23. april 2021.

Slik pipelinen er konstruert, lages det en ny modell for hver gang pipelinen kjøres. En mulig forbedring er å gi en trent modell som input til pipelinen for å kjøre pipelinen med en «varm start». Trainer-komponenten har et valgfritt parameter for modell,[10] så det vil være her man gjør det. Trainer kan også ta hyperparametere som input, noe som pipelinen ikke støtter i skrivende stund.

Pipelinen er laget slik at det tar csv-filer som datainput. Dette har vært tilstrekkelig for å utvikle og teste pipelinen. Teamet hos Statens vegvesen som jobber med trafikkdata som vi har vært i kontakt med, har også sin inndata som csv-filer, så dette har vært nyttig for dem. Derimot skal Saga-plattformen få mye data i *BigQuery* og bruke dette ekstensivt. Pipelinen bør også tilpasses dette for å fungere bedre som en del av Saga-plattformen. Skal pipelinen bruke *BigQuery* som input fremfor csv, må *CsvExampleGen*-komponenten erstattes med *BigQueryExampleGen*.[26] Bruken av *BigQueryExampleGen* krever også at en spørring blir gitt til `create_pipeline()` i `pipeline.py`. `kubeflow_runner.py` og eventuelt `local_runner.py` må også tilpasses for å gi dette parameteret når funksjonen kalles. Utover dette kreves ingen videre tilpasning av pipelinen for å bruke *BigQuery*.

Overvåkning av pipelinen og modellers ytelse har ikke blitt direkte utviklet av bachelorgruppen, men arbeidet med å kjøre pipelinens komponenter som jobber på GCP muliggjør dette. All logging av *Dataflow*-jobber og treningsjobber i *AI Platform* blir logget til GCP i *Logging*-verktøyet. Bachelorgruppen er ikke kjent med å skrive spørringer i dette verktøyet, men det vil herfra være mulig å hente ut data om det man lurer på angående pipelinens kjøring og modellenes ytelse.

Nok en mulig forbedring av pipelinen, vil være å kjøre den automatisk. Pipelinen kjøres nå manuelt ved å kjøre en TFX-kommando. Det kan for eksempel være ønskelig å kjøre pipelinen automatisk hver gang et visst antall

data har blitt lagt til i *BigQuery* siden sist kjøring av pipelinen. Eventuelt kan man lage en jobb som kjører pipelinen med et fast intervall. Etter gruppens forståelse av GCP, kan dette implementeres ved å lage en *Cloud Function*. Det skal være nok å konfigurere denne jobben til kjøre en enkelt TFX-kommando, gitt at Service Accounten som kjører jobben er autentisert.

6 Konklusjon

Bachelorgruppen fikk i oppgave å undersøke hvordan en maskinlæringspipeline kan se ut og hvordan den kan brukes hos Statens vegvesen. Produktet gruppen skulle jobbe mot var et 'Proof of Concept' av en slik pipeline. Sammen med oppdragsgiver fikk partene konkretisert en rekke behov som SVV hadde angående pipelinen.

Arbeidet som har gått med i driftsfasen av bacheloroppgaven har gått med til å realisere funksjonalitet som tilfredsstilte behovene, hvor samtlige behov har blitt tilfredsstilt.

Punktene under er behovene oppdragsgiver hadde. Disse er hentet fra forstudierapporten.[28]

- *Kunne kjøre ut en maskinlæringsmodell i GCP.*

  Pusher-komponenten i pipelinen kjører ut modeller til GCP.

- *Kunne kjøre ut nye versjoner av en maskinlæringsmodell i GCP.*

  Når Pusher-komponenten kjører ut en ny modell til GCP, blir den nye modellen satt som gjeldende i GCP, mens den gamle fortsatt er tilgjengelig.

- *Kunne eksponere en maskinlæringsmodell for input og bruke den fra andre applikasjoner.*

  Modeller som har blitt kjørt ut til GCP, ligger eksponert for input gjennom API-kall. Denne funksjonaliteten er innebygd i AI Platform på GCP.

- *Finne ut hvordan man kan overvåke en maskinlæringsmodell og dens evne til å predikere riktig.*

Det har blitt lagt fokus på å kjøre komponentene i pipelinen som jobber på GCP. Dette gjør at loggingen fra de forskjellige komponentene samles på ett sted, og at man gjennom loggingverktøyet i GCP blant annet kan overvåke maskinlæringsmodellens ytelse.

- *Finne ut hvordan man kan skalere en pipeline med hensyn på kapasitet og ytelse.*

  At orkestratoren til pipelinen kjører på Kubernetes, gjør det enkelt å gjøre om på clusterstørrelse. Kubernetes kan skaleres enkelt etter behov. En fordel med at komponentene i pipelinen kjører som jobber på GCP, er at disse automatisk kan skaleres med workers i *Dataflow*. Man kan manuelt sette antall workers og maksimum antall workers om man ønsker.

Pipelinen som er beskrevet i dette dokumentet svarer til de behovene oppdragsgiver har ønsket. Det 'Proof of Concept' som skulle utvikles har blitt utviklet, og bachelorgruppen leverer prosjektet videre til Statens vegvesen.

Referanser

[1] GitHub: *Bacheloroppgave for Statens vegvesen*, 2021.
Hentet fra:
`https://github.com/efdalen/bachelor2021`.
Lastet ned: 14. april 2021

[2] GitHub: *Bacheloroppgave for Statens vegvesen - Release Innlevering av bachelorarbeid*, 2021.
Hentet fra:
`https://github.com/efdalen/bachelor2021/releases/tag/1`.
Lastet ned: 11. mai 2021

[3] Google Cloud: *Global Locations - Regions & Zones*, 2021.
Hentet fra:
`https://cloud.google.com/about/locations/#europe`.
Lastet ned: 20. april 2021

[4] Google Cloud: *Specifying pipeline execution parameters*, 2021.
Hentet fra:
`https://cloud.google.com/dataflow/docs/guides/specifying-exec-params#setting-other-cloud-dataflow-pipeline-options`.
Lastet ned: 20. april 2021

[5] Google Cloud: *Introducing Cloud Dataflow Shuffle*, 2017.
Hentet fra:
`https://cloud.google.com/blog/products/gcp/introducing-cloud-dataflow-shuffle-for-up-to-5x-performance-improvement-in-`
Lastet ned: 20. april 2021

[6] Google Cloud: *Machine types*, 2021.
Hentet fra:
`https://cloud.google.com/compute/docs/machine-types`.
Lastet ned: 20. april 2021

[7] Tensorflow: *tfx.components.CsvExampleGen*, 2021.
Hentet fra:
`https://www.tensorflow.org/tfx/api_docs/python/tfx/components/`

CsvExampleGen.
Lastet ned: 20. april 2021

[8] Tensorflow: *tfx.components.SchemaGen*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/api_docs/python/tfx/components/SchemaGen.
Lastet ned: 20. april 2021

[9] Tensorflow: *tfx.components.Transform*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/api_docs/python/tfx/components/Transform.
Lastet ned: 20. april 2021

[10] Tensorflow: *tfx.components.Trainer*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/api_docs/python/tfx/components/Trainer.
Lastet ned: 20. april 2021

[11] Tensorflow: *Module: tfx.extensions.google_cloud_ai_platform.trainer.executor*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/api_docs/python/tfx/extensions/google_cloud_ai_platform/trainer/executor.
Lastet ned: 20. april 2021

[12] Tensorflow: *Creating a Custom TFX Executor*, 2019.
Hentet fra:
https://blog.tensorflow.org/2019/09/creating-custom-tfx-executor_19.html.
Lastet ned: 20. april 2021

[13] Tensorflow: *LatestBlessedModelResolver*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/api_docs/python/tfx/dsl/experimental/latest_blessed_model_resolver/

LatestBlessedModelResolver?hl=ru.
Lastet ned: 20. april 2021

[14] Tensorflow: *tfx.components.Evaluator*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/api_docs/python/tfx/components/Evaluator.
Lastet ned: 20. april 2021

[15] Github: *TFX Chicago Taxi Pipeline*, 2021.
Hentet fra:
https://github.com/tensorflow/tfx/tree/master/tfx/examples/chicago_taxi_pipeline.
Lastet ned: 20. april 2021

[16] Tensorflow: *Tensorflow Model Analysis*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/tutorials/model_analysis/tfma_basic.
Lastet ned: 20. april 2021

[17] Tensorflow: *tfx.components.Pusher*, 2021.
Hentet fra:
https://www.tensorflow.org/tfx/api_docs/python/tfx/components/Pusher.
Lastet ned: 20. april 2021

[18] V. Tatan, "Intro to ML Ops: Tensorflow Exntended (TFX)",towards datascience,Apr.2020.[Online]. Hentet fra:
https://towardsdatascience.com/intro-to-ml-ops-tensorflow-extended-tfx-39b6a
Lastet ned: 23.02.2021

[19] Argo Project, GitHub, *Argo Documentation*, 2021. Hentet fra:
https://argoproj.github.io/argo-workflows/.
Lastet ned: 01.03.2021

[20] Kubeflow, *Building Pipelines with the SDK*, 2020. Hentet fra:
https://www.kubeflow.org/docs/pipelines/sdk/.
Lastet ned: 01.03.2021.

[21] Google Cloud, *Projects*, 2021. Hentet fra:
`https://cloud.google.com/storage/docs/projects`.
Lastet ned: 05.03.2021

[22] Google Cloud, *Key terms*, 2021. Hentet fra:
`https://cloud.google.com/storage/docs/key-terms`.
Lastet ned: 22.04.2021

[23] Google Cloud, *VCP network overview*, 2021. Hentet fra:
`https://cloud.google.com/vpc/docs/vpc`.
Lastet ned: 22.04.2021

[24] Kubernetes, What is Kubernetes?, 2021. Hentet fra:
`https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`.
Lastet ned: 22.04.2021

[25] Google Cloud, *Google Kubernetes Engine Documentation*, 2021. Hentet fra:
`https://cloud.google.com/kubernetes-engine/docs`.
Lastet ned: 22.04.2021

[26] Tensorflow: *BigQueryExampleGen*, 2021.
Hentet fra:
`https://www.tensorflow.org/tfx/api_docs/python/tfx/extensions/google_cloud_big_query/example_gen/component/BigQueryExampleGen`.
Lastet ned: 26. april 2021

[27] Google Cloud: *Best practices for enterprise organizations*, 2021.
Hentet fra:
`https://cloud.google.com/docs/enterprise/best-practices-for-enterprise-organizations#project-structure`.
Lastet ned: 14. mai 2021

[28] J. A. Langholm, I. Andersen og E. F. Dalen, ”Forstudierapport”, *NTNU*, Februar 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[29] J. A. Langholm, I. Andersen og E. F. Dalen, ”Designrapport”, *NTNU*, Mars 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[30] J. A. Langholm, I. Andersen og E. F. Dalen, ”Ordbok”, *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[31] J. A. Langholm, I. Andersen og E. F. Dalen, ”README”, *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[32] Github: *Is there a way to disable cache...*, 2021.
Hentet fra:
`https://github.com/kubeflow/pipelines/issues/4857`.
Lastet ned: 19. mai 2021

7 Appendiks

A Gjennomgang av kode

Koden som blir fremvist under er hentet fra Github-repositoryet som bachelorgruppen har brukt under arbeidet med oppgaven.[1] Versjonen av koden som ligger i rapporten er den samme som fins i utgave *1 - Innlevering av bachelorarbeid* på GitHub.[2] En full kopi av utgaven kan lastes ned derfra.

Grunnet begrensninger i -biblioteket som anvendes i denne rapporten for å fremstille kode, kan det være mindre feil i linjenumre. Av og til har forfatter måttet tvinge linjeskift, og da blir dette registrert som en ny linje, noe som gjør at linjene under blir forskjøvet relativt til det kunstige linjeskiftet. Der de forklarende avsnittene refererer til konkrete linjenumre, er linjenumrene oppdatert til de linjenumre som står i rapporten. Avsnittene er ment til å leses sammen med kodeboksene i rapporten, og ikke med den faktiske kildekoden slik den fremstår på GitHub, da det enkelte steder vil være små forskjeller i linjenumre. Hver kodeboks' første linje er derimot synkron med kildekoden, og ikke alle kodebokser har kunstige linjeskift som skaper problemer.

A.1 template/pipeline/gcp_infrastructure.py

Prosjektgruppen har skrevet et skript for å sette opp nødvendig infrastruktur i GCP. Skriptet er skrevet i Python og benytter GCPs REST API-er og Pythonmoduler. Dette skriptet bør bare kjøres en gang, når man skal sette opp prosjektet i GCP, og helst samtidig som man gjennomgår README..[31]

```
1 import google.auth
2 from google.cloud import storage
3 from configs import GCS_BUCKET_NAME, PIPELINE_NAME,
  ↪  CLUSTER_SCALING
4 import cluster_creation
5 import network_creation
6 import json
7 import requests
```

```
8  import google.auth.transport.requests
9  import os
10 import subprocess
11 import sys
12 import time
```

Linje 1-12 inneholder importsetninger for å inkludere nødvendige Pythonmoduler og andre filer. Linje 4-5 (cluster_creating og network_creation) inneholder variabler av typen Dict, som underveis i skriptet oppdateres og konverteres til JSON for å nyttes som payload til REST API-ene.

```
14 # Get default project and authentication
15 try:
16     _, GOOGLE_CLOUD_PROJECT = google.auth.default()
17 except google.auth.exceptions.DefaultCredentialsError:
18     GOOGLE_CLOUD_PROJECT = ''
```

Linje 14-18 henter valgt prosjekts navn og et token for autentisering.

```
20 ### BUCKET CREATION ###
21 # Creates bucket with necessary subdirectories
22 print("Building GCS bucket...")
23 try:
24     client = storage.Client(project=GOOGLE_CLOUD_PROJECT)
25     try:
26         bucket = storage.Bucket(client, name=GCS_BUCKET_NAME)
27         bucket.create(location='eu')
28         print("Created bucket {} in project
           ↪  {}.".format(GCS_BUCKET_NAME, GOOGLE_CLOUD_PROJECT))
29     except Exception as e:
30         if e.code == 409:
31             print("Bucket already exists.")
32         else:
```

```
33              raise
34      try:
35          bucket = client.get_bucket(GCS_BUCKET_NAME)
36          path_modules = bucket.blob('input/modules/')
37          path_data = bucket.blob('input/data/')
38          path_output = bucket.blob('output/')
39          path_modules.upload_from_string('')
40          path_data.upload_from_string('')
41          path_output.upload_from_string('')
42          print("Built bucket hierarchy.")
43      except Exception as e:
44          print(e)
45  except Exception as e:
46      print(e)
```

Linje 20-46 omhandler opprettelse av en GCS-bucket med korrekt hierarki. Navn på bucket er definert i linje 26 og er hentet fra `configs.py`. Videre er feilhåndtering dersom bucketen allerede eksisterer. Linje 36-41 lager mappestrukturen i bucketen.

```
48  # Updloading necessary files to correct subdirectories
49  try:
50      client = storage.Client(project=GOOGLE_CLOUD_PROJECT)
51      bucket = client.get_bucket(GCS_BUCKET_NAME)
52      preprocessing = bucket.blob('input/modules/preprocessing.py')
53      trainer = bucket.blob('input/modules/trainer.py')
54      data = bucket.blob('input/data/data.csv')
55      preprocessing.upload_from_filename(
56          './modules/preprocessing.py'
57      )
58      print("Uploaded preprocessing.py to bucket
        ↪ {}/input/modules/preprocessing.py".format(
59          GCS_BUCKET_NAME
60      ))
```

```
61      trainer.upload_from_filename('./modules/trainer.py')
62      print("Uploaded trainer.py to bucket
        ↪  {}/input/modules/trainer.py".format(GCS_BUCKET_NAME))
63      data.upload_from_filename('./data/data.csv')
64      print("Uploaded data to bucket
        ↪  {}/input/data/data.csv".format(GCS_BUCKET_NAME))
65  except Exception as e:
66      print(e)
```

Linje 50-63 sørger for å laste opp nødvendige filer og data til GCS for bruk av pipelinen. Her lastes kode for preprosessering og trening opp, samt dataen som skal brukes til trening og testing.

```
64  ### Network creation ###
65  POST_URL = "https://www.googleapis.com/compute/v1/projects/{}
    ↪  /global/networks".format(GOOGLE_CLOUD_PROJECT)
66  POST_PAYLOAD = network_creation.network
67  POST_PAYLOAD["selfLink"] = "projects/{}
    ↪  /global/networks/default".format(GOOGLE_CLOUD_PROJECT)
68  POST_PAYLOAD = json.dumps(POST_PAYLOAD)
69
70  # Creating credential token
71  creds, project = google.auth.default(scopes=[
72      "https://www.googleapis.com/auth/cloud-platform"
73  ])
74  auth_req = google.auth.transport.requests.Request()
75  creds.refresh(auth_req)
76
77  # REST API call to create network
78  print("Building network...")
79  response = requests.post(POST_URL, data = POST_PAYLOAD,
    ↪  headers={'Authorization': 'Bearer {}'.format(creds.token)})
80  resp_json = json.loads(response.text)
81
```

```
82 # Catching HTTP errors
83 if response.status_code != 200:
84     if resp_json["error"]["errors"][0]["reason"] ==
       ↪  "alreadyExists":
85         print("Network already exists.")
86     else:
87         print("Network creation failed. \n")
88         print(response.status_code)
89         print(response.text)
90 else:
91     print("Network is deploying...")
```

Linje 65-80 omhandler oppretting av et VPC-nettverk. Her brukes et REST API, som kalles med en POST-forespørsel bestående av JSON-data om nettverkskonfigurasjon. I linje 67 sikres hvilket prosjekt nettverket opprettes i. Linje 71-75 oppdaterer et token, nødvendig for autentisering av forespørselen.

```
91 ### CLUSTER CREATION ###
92 POST_PAYLOAD = {}
93 POST_URL = ""
94
95 # Defines which JSON data to POST to which URL
96 if CLUSTER_SCALING == 'standard':
97     POST_PAYLOAD = cluster_creation.standard
98     POST_URL =
       ↪  "https://container.googleapis.com/v1beta1/projects/{}
       ↪  /zones/europe-west3-a/clusters
       ↪  ".format(GOOGLE_CLOUD_PROJECT)
99 else:
100    POST_PAYLOAD = cluster_creation.autopilot
101    POST_URL = "https://container.googleapis.com/v1/projects/{}
       ↪  /locations/europe-west3/clusters
       ↪  ".format(GOOGLE_CLOUD_PROJECT)
```

```
102
103 # Set variables for cluster creation
104 CLUSTER_NAME = PIPELINE_NAME + '-cluster'
105 CLUSTER_NAME = CLUSTER_NAME.replace('_','-')
106
107 # Alters payload to match GCP project and adds relevant cluster
    ↪ name
108 POST_PAYLOAD["cluster"]["name"] = CLUSTER_NAME
109 POST_PAYLOAD["cluster"]["network"] =
    ↪ "projects/{}/global/networks/default
    ↪ ".format(GOOGLE_CLOUD_PROJECT)
110 POST_PAYLOAD["cluster"]["subnetwork"] =
    ↪ "projects/{}/regions/europe-west3/subnetworks/default
    ↪ ".format(GOOGLE_CLOUD_PROJECT)
111 # Dict to JSON for POST data type
112 POST_PAYLOAD = json.dumps(POST_PAYLOAD)
113
114 # Creating credential token
115 creds, project = google.auth.default(scopes=["
    ↪ https://www.googleapis.com/auth/cloud-platform"])
116 auth_req = google.auth.transport.requests.Request()
117 creds.refresh(auth_req)
118
119 # REST API call to build cluster
120 print("Building cluster...")
121 response = requests.post(POST_URL, data = POST_PAYLOAD,
    ↪ headers={'Authorization': 'Bearer {}'.format(creds.token)})
122 resp_json = json.loads(response.text)
123
124 # Waiting for network to be fully configured
125 while response.status_code == 400:
126     response = requests.post(POST_URL, data = POST_PAYLOAD,
      ↪ headers={'Authorization': 'Bearer
      ↪ {}'.format(creds.token)})
```

```
127     resp_json = json.loads(response.text)
128 
129 # Catching HTTP errors when creating cluster
130 if response.status_code != 200:
131     if resp_json["error"]["status"] == "ALREADY_EXISTS":
132         print("Cluster already exists.")
133     else:
134         print("Cluster build failed. \n")
135         print(response.status_code)
136         print(response.text)
137 else:
138     print("Cluster build completed successfully. Cluster
      ↪  deployment in progress. Continue with KFP deployment.")
```

Linje 92-127 sørger for å opprette et cluster i Kubernetes Engine. Dette er nødvendig for å rulle ut Kubeflow Pipelines i GCP. Linje 96-101 er til dags dato ikke brukbar, da skaleringskonfigurasjon av clusteret må være standard for å være kompatibelt med Kubeflow Pipelines. Kodesnutten ligger likevel inne for at man i fremtiden kan nytte automatisk skalering ved behov. Linje 104-105 definerer navnet på clusteret og sørger for at understreker byttes ut med bindestreker, da det er eneste tillatte spesialtegn. Linje 108-111 sørger for at clusteret blir konfigurert riktig, ved å oppdatere payloaden. Linje 121 utfører POST-forespørselen med nødvendig payload. Gjennom skriptet tar det kort tid fra forsepørselen om å opprette nettverk til å deploye cluster skjer. Det er derfor sannsynlig at nettverket ikke er ferdig konfigurert før clusteret forsøkes å deployes. Derfor sørger linje 125-127 for å vente til nettverket er ferdig konfigurert før clusteret deployes. Resten av koden er feilhåndtering.

A.2 template/pipeline/pipeline.py

Denne filen definerer definerer strukturen og flyten til pipelinen. Komponentene blir en etter en definert med konfigurasjon importert fra *configs.py* og med outputen til tidligere komponenter som parametere. En forklaring av flyten og strukturen til pipelinen finnes i kapittel 3 og i

designrapporten.[29]

Begynnelsen av filen inneholder importsetninger av biblioteker som senere brukes i filen.

```
54 def create_pipeline(
55     pipeline_name: Text,
56     pipeline_root: Text,
57     metadata_connection_config:
       ↪  Optional[metadata_store_pb2.ConnectionConfig] = None,
58     beam_pipeline_args: Optional[List[Text]] = None,
59     enable_cache: bool = False,
60 
61     csv_example_gen_args: Optional[Dict[Text, Any]] = None,
62     schema_gen_args: Optional[Dict[Text, Any]] = None,
63     transform_args: Optional[Dict[Text, Any]] = None,
64     trainer_args: Optional[Dict[Text, Any]] = None,
65     model_resolver_args: Optional[Dict[Text, Any]] = None,
66     evaluator_args: Optional[Dict[Text, Any]] = None,
67     pusher_args: Optional[Dict[Text, Any]] = None,
68 ) -> pipeline.Pipeline:
```

Dette er alle parametrene som gis til selve funksjonen som definerer pipelinen. Disse defineres i `configs.py` og gis via `kubeflow_runner.py`.

```
54     components = []
```

Denne listen vil bli fylt med de individuelle TFX-komponentene som defineres senere i filen.

```
72     # Brings data into the pipeline or otherwise joins/converts
       ↪  training data.
73     example_gen = CsvExampleGen(**csv_example_gen_args)
74     components.append(example_gen)
```

Dette er den første komponenten som defineres i pipelinen. Om man heller vil bruke BigQuery som input, må man erstatte linje 73 og heller bruke `bigqueryexamplegen(query=query)` der `query` er en streng som er spørringen som skal gjøres mot bigquery. Utover dette kreves ingen andre tilpasninger i pipelinen for å bruke BigQuery fremfor csv.

```
76      # Computes statistics over data for visualization and example
        ↪ validation.
77      statistics_gen =
        ↪ StatisticsGen(examples=example_gen.outputs['examples'])
78      components.append(statistics_gen)
```

Genererer statistikk over inputdataen. Dette brukes senere for å validere dataen. Man kan også bruke output herfra for å analysere dataen.

```
80      # Generates schema based on statistics files.
81      schema_gen = SchemaGen(
82          statistics=statistics_gen.outputs['statistics'],
83          **schema_gen_args)
84      components.append(schema_gen)
```

Lager et skjema for dataen. Skjemaet beskriver dataen med typer, kategorier og rammer. Skjemaet er automatisk generert og stemmer ikke nødvendigvis 100%, så dataviter bør se over og eventuelt modifisere. Skjemaet kan skrives helt manuelt, men det er gjerne greiest å modifisere et som kommer fra denne komponenten slik man vil ha det. Skjemaet er å finne som klartekst i filen `<pipeline_root>/SchemaGen/schema/<artifact_id>/schema.pbtxt`. Ved senere kjøring kan det være ønskelig å heller bruke et skjema man har laget selv. Det automatisk genererte skjemaet er bare gjetning fra en datamaskin, og hvis man skal forsikre seg om at man har et korrekt skjema, bør man ha laget det selv slik at det ikke oppstår problemer en dag man møter på søppeldata. Dette er ikke funksjonalitet som bachelorgruppen har prioritert å utvikle, så slik pipelinen står i skrivende stund, genereres skjema automatisk hver gang man kjører pipelinen.

```
86      # Performs anomaly detection based on statistics and data
        ↪ schema.
87      example_validator = ExampleValidator(
88          statistics=statistics_gen.outputs['statistics'],
89          schema=schema_gen.outputs['schema'])
90      components.append(example_validator)
```

Validerer at inndataen passer til skjemaet og oppdager eventuelle uregelmessigheter. Komponenten kan kjenne igjen «skew» mellom treningsdata og evalueringsdata og data drift. Om man ikke får rensket dataen, vil det bli problemer når man senere skal trene modellen.

```
92      # Performs transformations and feature engineering in
        ↪ training and serving.
93      transform = Transform(
94          examples=example_gen.outputs['examples'],
95          schema=schema_gen.outputs['schema'],
96          **transform_args)
97      components.append(transform)
```

Transform utfører førprosessering på dataen. Transform trenger en `preprocessing_fn()`. Denne må defineres i en Python-modul som så gis til Transform og brukes i førprosesseringen. Konfigurasjon av denne komponenten må gjøres av dataviter som kjenner modellen og vet hvilke krav den stiller til inndata og formen på de.

```
99      trainer = Trainer(
100         examples=transform.outputs['transformed_examples'],
101         transform_graph=transform.outputs['transform_graph'],
102         schema=schema_gen.outputs['schema'],
103         **trainer_args
104     )
105     components.append(trainer)
```

Kjører selve treningen av modellen. Dette må konfigureres av dataviter på lik linje med Transform, da denne er spesifikk for modellen. Denne kan også ta en ferdig modell som input for å kjøre «varm start» ved å sette `base_model`-parameteret. Dette gjør man hvis man ikke ønsker å trene en modell helt fra «scratch».

```
107     # Get the latest blessed model for model validation.
108     model_resolver = ResolverNode(**model_resolver_args)
109     components.append(model_resolver)
```

Henter den for øyeblikket gjeldende versjonen av modellen fra *Cloud Storage*, slik at den kan sammenlignes med den nye fra Trainer.

```
111     evaluator = Evaluator(
112         examples=example_gen.outputs['examples'],
113         model=trainer.outputs['model'],
114         baseline_model=model_resolver.outputs['model'],
115         **evaluator_args,
116     )
117     components.append(evaluator)
```

Evaluator sammenligner den nye modellen med den gamle. Den som er best blir gitt videre slik at den kan sendes til endepunkt. Evaluator behøver ikke konfigurasjon, og vil uten konfigurasjon bare sammenligne total ytelse på de to. Med konfigurasjon kan man be komponenten se på spesielle tilfeller.

```
119     # Checks whether the model passed the validation steps and
        ↪  pushes the model
120     # to a file destination if check passed.
121     pusher = Pusher(
122         model=trainer.outputs['model'],
123         model_blessing=evaluator.outputs['blessing'],
124         **pusher_args,
125     )
126     components.append(pusher)
```

Pusher sender modellen til endepunkt.

```
128     return pipeline.Pipeline(
129         pipeline_name=pipeline_name,
130         pipeline_root=pipeline_root,
131         components=components,
132         enable_cache=enable_cache,
133         metadata_connection_config=metadata_connection_config,
134         beam_pipeline_args=beam_pipeline_args,
135     )
```

Disse siste linjene i `pipeline.py` returnerer pipelinen til `kubeflow_runner.py` som kan ses på som hovedfilen. Pipeline-objektet blir kompilert og siden gitt til Kubeflow Pipelines der den kjører.

A.3 template/pipeline/configs.py

Den forrige filen definerte bare sammenhengen mellom de forskjellige TFX-komponentene og var veldig vag i hvordan disse konkret skulle fungere. All konfigurasjon av komponentene samt øvrig konfigurasjon ligger i filen `template/pipeline/configs.py`. Et viktig unntak er skriptene som kjøres av Transform- og Trainer-komponentene.

Linje 1 til 41 inneholder importsetninger av biblioteker som blir brukt senere.

```
42 # Navnet brukes for å identifisere denne pipelinen. Navnet brukes
    ↪  konkret i KFP og i GCS.
43 PIPELINE_NAME = 'dev-pipeline'
```

Som kommentaren sier, brukes dette navnet for å identifisere pipelinen. Dette vil bli brukt senere i konfigurasjonsfilen, og også av `pipeline.py` når pipelinen kompileres.

```
45 # Brukes for å definere config for cluster.
46 # 'standard' or 'autopilot'
47 CLUSTER_SCALING = 'standard' # Kan også bruke autopilot, sett da
   ↪ "autopilot"
```

Definerer skaleringen av Kubernetes-clusteret. Ved standard, skalerer man clusteret manuelt, mens autopilot skalerer automatisk. Det er per nå ikke støtte for kjøring av Kubeflow Pipelines på autopilot-cluster da autopilot er under etablering hos GCP.

```
49 # GCP related configs.
50
51 # Following code will retrieve your GCP project. You can choose
   ↪ which project
52 # to use by setting GOOGLE_CLOUD_PROJECT environment variable.
53 try:
54   import google.auth  # pylint: disable=g-import-not-at-top
55   try:
56     _, GOOGLE_CLOUD_PROJECT = google.auth.default()
57   except google.auth.exceptions.DefaultCredentialsError:
58     GOOGLE_CLOUD_PROJECT = ''
59 except ImportError:
60   GOOGLE_CLOUD_PROJECT = ''
```

Denne kodesnutten henter navnet på GCP-prosjektet slik at man slipper å skrive dette selv.

```
62 # Specify your GCS bucket name here. You have to use GCS to store
   ↪ output files
63 # when running a pipeline with Kubeflow Pipeline on GCP or when
   ↪ running a job
64 # using Dataflow. Default is
   ↪ '<gcp_project_name>-kubeflowpipelines-default'.
65 # This bucket is created automatically when you deploy KFP from
   ↪ marketplace.
```

```
66
67  # Using pipeline name to name GCS bucket
68  bucket_name = PIPELINE_NAME
69
70  # String to lowercase
71  bucket_name = bucket_name.lower()
72
73  # For norwegian letters, use ae, o, a
74  bucket_name = bucket_name.replace('æ', 'ae')
75  bucket_name = bucket_name.replace('ø', 'o')
76  bucket_name = bucket_name.replace('å', 'a')
77
78  # Removing all characters but a-z, 0-9, whitespace, '_', '-' and
    ↪  '.'
79  bucket_name = re.sub("[^a-z0-9\s\_\-\.]+", "", bucket_name)
80
81  # Swapping whitespace with '-'
82  bucket_name = re.sub("\s", "-", bucket_name)
83
84  # Remove repeating '-' (dashes) and dash after '.' and '_'
85  iter = list(bucket_name)
86  bucket_name = ''
87  for i in range (0,len(iter)):
88      if iter[i] == '-' and iter[i-1] in '.-_':
89          iter[i] = ''
90      bucket_name += str(iter[i])
91
92  GCS_BUCKET_NAME = GOOGLE_CLOUD_PROJECT + '-' + bucket_name
```

Et navn på bucketen avledes fra pipelinenavnet. Eventuelle spesialtegn som ikke kan være i et bucketnavn lukes ut slik at det endelige bucketnavnet er på riktig format.

```
94  # TFX pipeline produces many output files and metadata. All
    ↪  output data will be
95  # stored under this OUTPUT_DIR.
96  OUTPUT_DIR = os.path.join('gs://', GCS_BUCKET_NAME, 'output')
97  INPUT_DIR = os.path.join('gs://', GCS_BUCKET_NAME, 'input')
98
99  # The last component of the pipeline, "Pusher" will produce
    ↪  serving model under
100 # SERVING_MODEL_DIR.
101 SERVING_MODEL_DIR = os.path.join(OUTPUT_DIR, 'serving_model')
102
103 # Specifies data file directory. DATA_PATH should be a directory
    ↪  containing CSV
104 # files for CsvExampleGen in this example. By default, data files
    ↪  are in the
105 # GCS path: `gs://{GCS_BUCKET_NAME}/input/data/`.
106
107 DATA_PATH = os.path.join(INPUT_DIR, 'data')
108 PREPROCESSING_PATH = os.path.join(INPUT_DIR,
    ↪  'modules/preprocessing.py')
109 TRAINER_PATH = os.path.join(INPUT_DIR, 'modules/trainer.py')
```

Diverse stier som brukes i pipelinen defineres her; både for input og output. Stiene for preprocessing og trainer peker på konfigurasjonen for disse to komponentene. Det er disse filene som må skrives av dataviteren og være tilpasset modellen som skal kjøre i pipelinen.

Spesifikt for oppgavens proof of concept, blir filene som skal ligge i disse stiene lastet opp av skriptet `gcp_infrastructure.py`

```
111 # Setter region til europa. Pga begrensninger i GCP, kan vi ikke
    ↪  kjøre alle tjenester i samme region. Deploying av modeller
    ↪  vil skje i europe-west1, mens resten skjer i europe-west4.
112 GOOGLE_CLOUD_REGION = 'europe-west4'
```

```
113 PUSHER_REGION = 'europe-west1'
```

De forskjellige regionene hvor pipelinen kjører. Ingen region i Europa støtter all funksjonaliteten alene,[3] så pipelinen kjører per dags dato både i Belgia (europe-west1) og i Holland (europe-west4).

```
115 ENABLE_CACHE = False
```

Skrur av mellomlagring av komponenters output. Om en komponent har blitt kjørt tidligere med identisk input, blir den ikke kjørt igjen, og outputen fra den tidligere kjøringen blir heller brukt. Dette kan man også skru av for KFP. Det har oppstått problemer med doble artefakter, så dette er avskrudd. Dette innebærer at enkelte komponenter fikk dobbel input, og ikke kunne kjøre. Et problem med Kubernetes gjorde at bachelorgruppen uheldigvis måtte skru dette av i KFP.[32] Det vil være fordelaktig å komme tilbake til dette senere for å rette opp i feilen.

```
117 #Argumenter til dataflow
118 DATAFLOW_BEAM_PIPELINE_ARGS = [
119     '--project=' + GOOGLE_CLOUD_PROJECT,
120     '--runner=DataflowRunner',
121     '--temp_location=' + os.path.join('gs://', GCS_BUCKET_NAME,
    ↪  'tmp'),
122     '--region=' + GOOGLE_CLOUD_REGION,
123 
124     # Temporary overrides of defaults.
125     '--disk_size_gb=50',
126     '--experiments=shuffle_mode=auto',
127     '--machine_type=e2-standard-8',
128     ]
```

Argumenter som gis til Dataflow på GCP.[4]

- `--project` sier hvilket prosjekt det gjelder.

- `--runner` sier hvilken runner som skal brukes. `DataflowRunner` bør helst brukes. Alternativet er `Directrunner` som vil være å kjøre lokalt.
- `--temp_location` sier hvor midlertidige filer skal lagres.
- `--region` sier hvilken GCP-region Dataflow skal kjøre.
- `--disk-size` angir diskstørrelsen til hver Compute Engine worker.
- `--experiments=shuffle_mode` angir bruken av Dataflow Shuffle, som er en optimalisering av Dataflowjobber.[5]
- `--machine-type` angir VM-typen for Compute Engine workers.[6]

Det er andre parametere man kan spesifisere her; blant annet antall workers man vil ha til jobben.[4]

```
130 #Trening på GCP
131 GCP_AI_PLATFORM_TRAINING_ARGS = {
132     'project': GOOGLE_CLOUD_PROJECT,
133     'region': GOOGLE_CLOUD_REGION,
134 }
```

Oppgir prosjekt og region for trening som skjer på AI Platform i GCP.

```
136 #Serving på GCP
137 GCP_AI_PLATFORM_SERVING_ARGS = {
138     'model_name': PIPELINE_NAME.replace('-','_'),  # '-' is not
    ↪  allowed.
139     'project_id': GOOGLE_CLOUD_PROJECT,
140     'regions': [PUSHER_REGION],
141 }
```

Argumenter til serving av modeller på GCP. Modeller gis det samme navnet som pipelinen for ryddighet. Modeller pushes til Belgia (europe-west1), da det ikke støttes i Holland (europe-west4).

```
#Parametere til TFX-komponenter
```

Videre vil det komme konfigurasjon av de forskjellige komponentene som er beskrevet i `pipeline.py` i kapittel A.2. Konfigurasjonen er i form av objekter som inneholder parametere til de forskjellige funksjonene som definerer TFX-komponentene. Merk at objektene ikke inneholder de paramtere som er output fra andre komponenter.

A.3.0.1 CsvExampleGen

```
CSV_EXAMPLE_GEN_ARGS = {
    'input_base': DATA_PATH,
}
```

Argumenter til CsvExampleGen.[7] Denne komponenten behøver bare en sti. `input_base` er stien til en mappe der det ligger en eller flere csv-filer.

A.3.0.2 SchemaGen

```
SCHEMA_GEN_ARGS = {
    'infer_feature_shape': True,
}
```

Argumenter til SchemaGen.[8] `infer_feature_shape` angir hvorvidt komponenten skal legge inn formene til dataens attributter automatisk, slik at man ikke trenger å definere skjemaet selv.

A.3.0.3 Transform

```
TRANSFORM_ARGS = {
    #sti til konfigurasjonsfil for transform (foerprosessering)
    'module_file': PREPROCESSING_PATH,
}
```

Argumenter til Transform.[9] `module_file` er stien til en fil som må inneholde en funksjon som heter `preprocessing_fn()`. Man kan se på denne funksjonen som hovedfunksjonen til førprosesseringen. Funksjonen må ha denne signaturen:

```
def preprocessing_fn(inputs: Dict[Text, Any]) -> Dict[Text, Any]:
```

Typen til både inputdicten og outputdicten må være av `tf.Tensor` eller `tf.SparseTensor`
Modulfilen som inneholder denne funksjonen skal ligge i en GCS bucket med stien som defineres på linje 108.

A.3.0.4 Trainer

```
158 #treningssteg
159 TRAIN_NUM_STEPS = 100
160
161 #evalueringssteg
162 TRAINER_EVAL_NUM_STEPS = 15
163
164 TRAINER_ARGS = {
165     #sti til konfigurasjonsfil for trainer
166     'module_file': TRAINER_PATH,
167
168     #får trening til å skje på GCP
169     'custom_executor_spec': executor_spec.ExecutorClassSpec(
        ↪  ai_platform_trainer_executor.GenericExecutor),
170     'custom_config': {
171         ai_platform_trainer_executor.TRAINING_ARGS_KEY:
            ↪  GCP_AI_PLATFORM_TRAINING_ARGS
172     },
173     'train_args':
        ↪  trainer_pb2.TrainArgs(num_steps=TRAIN_NUM_STEPS),
174     'eval_args':
        ↪  trainer_pb2.EvalArgs(num_steps=TRAINER_EVAL_NUM_STEPS)
```

```
175 }
```

Konfigurasjon av treningen.[10]

- `TRAIN_NUM_STEPS` angir antall steg i treningen. Dette blir siden gitt på linje 173.
- `TRAINER_EVAL_NUM_STEPS` angir antall steg i evalueringen i treningen. Dette blir siden gitt på linje 174.
- `module_file` er stien til filen som definerer treningen. Når man bruker `GenericExecutor`, slik som pipelinen er definert, må filen ha en funksjon som heter `run_fn()` med denne signaturen:

  ```
  def run_fn(trainer.fn_args_utils.FnArgs)
  ```

  Den trente modellen må lagres til stien i `FnArgs.serving_model_dir`.
- `custom_executor_spec` spesifieserer hvilken executor som skal utføre treningen. Her er den definert med `GenericExecutor`. Man har to valg her.[11] Eventuelt kan man lage sin egen.[12]
- `custom_config` inneholder prosjekt og region slik de ble definert på linje 132 og 133.

A.3.0.5 ResolverNode

```
178 MODEL_RESOLVER_ARGS = {
179     'instance_name': 'latest_blessed_model_resolver',
180     'resolver_class':
        ↪  latest_blessed_model_resolver.LatestBlessedModelResolver,
181     'model': Channel(type=Model),
182     'model_blessing': Channel(type=ModelBlessing),
183 }
```

Argumenter for `ResolverNode` som henter gjeldende modell.[13]

A.3.0.6 Evaluator

```
185 # Change this value according to your use cases.
186 EVAL_ACCURACY_THRESHOLD = 0.6
187
188 # Uses TFMA to compute a evaluation statistics over features of a
    ↪ model and
189 # perform quality validation of a candidate model (compared to a
    ↪ baseline).
190 EVAL_CONFIG = tfma.EvalConfig(
191     model_specs=[
192         # This assumes a serving model with signature
            ↪ 'serving_default'. If
193         # using estimator based EvalSavedModel, add
            ↪ signature_name: 'eval' and
194         # remove the label_key.
195         tfma.ModelSpec(label_key='tips')
196     ],
197     metrics_specs=[
198         tfma.MetricsSpec(
199             # The metrics added here are in addition to those
                ↪ saved with the
200             # model (assuming either a keras model or
                ↪ EvalSavedModel is used).
201             # Any metrics added into the saved model (for example
                ↪ using
202             # model.compile(..., metrics=[...]), etc) will be
                ↪ computed
203             # automatically.
204             # To add validation thresholds for metrics saved with
                ↪ the model,
205             # add them keyed by metric name to the thresholds
                ↪ map.
206             metrics=[
207                 tfma.MetricConfig(class_name='ExampleCount'),
```

```
208                 tfma.MetricConfig(class_name='BinaryAccuracy',
209                   threshold=tfma.MetricThreshold(
210                       value_threshold=tfma.GenericValueThreshold(
211                           lower_bound={'value':
                                ↪ EVAL_ACCURACY_THRESHOLD}),
212                       change_threshold=
                            ↪ tfma.GenericChangeThreshold(
213                           direction=tfma.MetricDirection.
                                ↪ HIGHER_IS_BETTER,
214                           absolute={'value': -1e-10})))
215             ]
216         )
217     ],
218     slicing_specs=[
219         # An empty slice spec means the overall slice, i.e. the
            ↪ whole dataset.
220         tfma.SlicingSpec(),
221         # Data can be sliced along a feature column. In this
            ↪ case, data is
222         # sliced along feature column trip_start_hour.
223         tfma.SlicingSpec(feature_keys=['trip_start_hour'])
224     ]
225 )
226
227 EVALUATOR_ARGS = {
228     'eval_config': EVAL_CONFIG,
229 }
```

Konfigurasjon av `Evaluator`.[14] Konfigurasjonen her er et eksempel og er hentet fra TFXs taxi-eksempel.[15] Konfigurasjonen her bruker i stor grad Tensorflow Model Analysis.[16] Denne konfigurasjonen er spesifikk for modellen, og må utvikles av dataviter.
`EVAL_ACCURACY_THRESHOLD` angir terskelverdi for minste akseptable nøyaktighet.

A.3.0.7 Pusher

```
231 PUSHER_ARGS = {
232     'push_destination': pusher_pb2.PushDestination(
233         filesystem=pusher_pb2.PushDestination.Filesystem(
234             base_directory=SERVING_MODEL_DIR
235         )
236     ),
237     'custom_executor_spec': executor_spec.ExecutorClassSpec(
        ↪ ai_platform_pusher_executor.Executor),
238     'custom_config': {
239         #får serving til å skje på GCP
240             ai_platform_pusher_executor.SERVING_ARGS_KEY:
241                 GCP_AI_PLATFORM_SERVING_ARGS
242         },
243 }
```

Argumenter til `Pusher`.[17] Denne er konfigurert til å både lagre modellen i en bucket, samt deploye den til AI Platform Models, der den er er klar for eksponering mot tredjepartsapplikasjoner.

- `push_destination` angir stien i GCS der modellen skal lagres. Denne stien er definert på linje 101.
- `custom_executor_spec` definerer hvilken executor som skal kjøre pushingen. Her er denne konfigurert med en som pusher til GCP AI Platform.
- `custom_config` inneholder konfigurasjonen fra linje 137 som inneholder modellnavn, prosjekt og region.

4 Sluttrapport

MLOps på Google Cloud Platform
Sluttrapport

Ingvild Andersen, Jon Akselberg Langholm, Erik Flæsen Dalen

19. mai 2021

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
18.05.2021	1.0	Første versjon av sluttrapport	Erik F.D., Ingvild A. og Jon L.
20.05.2021	1.1	Språkvask, mindre revideringer etter tilbakemelding fra veileder	Erik F.D., Ingvild A. og Jon L.

Innhold

1 Forord **5**

2 Innledning **6**
2.1 Dokumentets hensikt 6
2.2 Definisjoner og forkortelser 6
2.3 Oversikt over dokumentet 7

3 Oppgavebeskrivelse **7**

4 Hvordan ble oppgaven løst **8**
4.1 Litteratur og Internett 8
4.2 Metoder og standarder 8
4.3 Maskinvare 9
4.4 Programvare 9
4.5 Arbeidsfordeling 10
4.6 Dokumentasjon 11

5 Gjennomføring av prosjektet **12**
5.1 Prosess 12
5.1.1 Oppstart 12
5.1.2 Utførelse 14
5.1.3 Ferdigstilling 17
5.1.4 Planlagt og faktisk arbeidsprosess 17
5.2 Måloppnåelse 18
5.2.1 Effektmål 19
5.2.2 Resultatmål 19
5.2.3 Prosessmål 20
5.3 Timeregnskap 21
5.4 Refleksjon rundt prosessen 21
5.4.1 Suksessfaktorer og risikovurdering 23
5.5 Endringer 24

6 Videre arbeid **24**

Referanser **25**

Figurer

2 Enkel oversikt over oppgaven. 8
3 Struktur i zip-filen oppgaven leveres som 12
4 Stipulert arbeidsprosess 18
5 Faktisk arbeidsprosess 18

1 Forord

Denne sluttrapporten skrives i sammenheng med bacheloroppgaven «MLOps på Google Cloud Platform», forfattet av Erik F. Dalen, Jon Langholm og Ingvild Andersen. Den er den fjerde av fire rapporter produsert i forbindelse med faget IDRI3001 *Bacheloroppgave i Informatikk, drift av datasystemer*". Hensikten er å presentere leser for hvordan prosjektperioden har utfoldet seg, samt erfaringer prosjektgruppen sitter igjen med.

Oppgavens hensikt var at prosjektgruppen skulle lære hvordan man gjennomfører et prosjekt på vegne av en reell arbeidsgiver, samt de tilpasningene som behøves for å møte bedriftens mål og krav. Prosjektgruppen skulle kombinere kunnskap fra studiet med ny kunnskap for å løse bedriftens problemstilling. Fra prosjektgruppens side var det også et ønske å øke medlemmenes kompetanse i maskinlæring og maskinlæringsverktøy i Google Cloud Platform. Oppgaven har vært faglig krevende, og gruppen har måttet tilegne seg mye ny kunnskap for å kunne møte bedriftens krav. Dette innebar å sette seg inn i hvordan maskinlæringsmodeller fungerer, samt hvilke produksjonsbehov disse har for å kunne fungere optimalt i skyplattformen. I tillegg til økt kompetanse i bruk av Google Cloud Platform og maskinlæringsverktøy, endte gruppen også opp med ny erfaring i bruk av TensorFlow Extended, Kubeflow og Kubernetes.

Både oppgaven og resultatet hadde ikke blitt til uten hjelp fra Statens vegvesen, konsulenter hos Bekk og veileder hos NTNU. Vi ønsker derfor å rette en takk til alle de involverte.

Torgeir Thoresen (BEKK)
Vår veileder på vegne av Statens vegvesen. Takk for tett oppfølging, veiledning, tilbakemelding på rapporter og oppklaring rundt bedriftens krav og ønsker. Du har hatt en stor positiv påvirkning i prosessen.

Safurudin Mahic (BEKK)
Sensor på vegne av Statens vegvesen. Takk for all veiledning, og alle møter underveis i prosessen. Vi setter også stor pris på at du tar deg tid til å sensurere oppgaven på vegne av oppdragsgiver.

Jostein Lund (NTNU)

Veileder og vår universitetslektor hos NTNU. Takk for god oppfølging og veiledning, både gjennom dette bachelorprosjektet og de siste tre årene med skolegang.

Stein Meisingseth (NTNU)
Vår universitetslektor hos NTNU. Oppgaven hadde ikke blitt til uten din hjelp. Vi vil også takke for god veiledning og støtte gjennom de tre siste årene med skolegang.

Lars Meisingseth (SVV)
Produkteier på vegne av Statens vegvesen. Takk for at du ga oss sjansen til å skrive hos dere. Vi har også satt pris på veiledningen vi fikk i startfasen, og at du har fulgt opp underveis.

Øvrig
Det har vært flere kloke hoder som har vært innom prosessen og som har kommet med gode råd og veiledning. Takk til alle som har vist interesse i Statens vegvesen, og maskinlæringsgruppa hos Bekk som har bidratt når vi har stått fast.

2 Innledning

2.1 Dokumentets hensikt

Dokumentets hensikt er å konkludere og reflektere rundt bachelorprosessen i sin helhet. Det skal gå inn på hvordan oppgaven ble løst, og hvilke verktøy som ble tatt i bruk. Den informerer leser om hvordan prosjektgruppen har fordelt arbeidet, og hvordan prosjektet ble gjennomført. Dette innebærer informasjon om tidsbruk og refleksjon rundt hvordan prosessen ble lagt opp. Videre sammenstilles målene som ble satt i forstudien mot reell måloppnåelse, samt endringer som har blitt gjort i ettertid.

2.2 Definisjoner og forkortelser

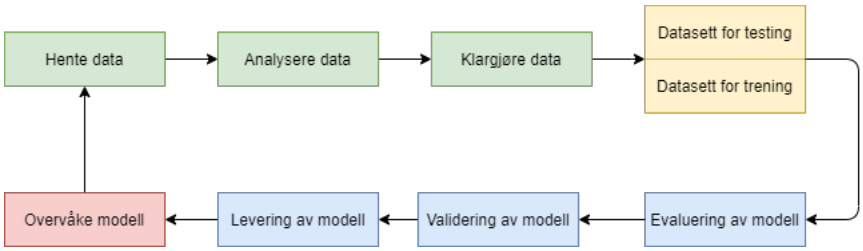
Begreper og forkortelser er samlet inn i en felles prosjektordbok.[11]

2.3 Oversikt over dokumentet

Første del av dokumentet (2) beskriver dokumentets hensikt (2.1), hvor man finner definisjoner og forkortelser (2.2) og gir en overordnet oversikt i dokumentstrukturen (2.3). Neste kapittel gir en beskrivelse av oppgaven (3), før leseren presenteres med hvordan denne har blitt løst (4). Helt konkret innebærer dette bruk av litteratur og internettsøk (4.1), metoder og standarder (4.2), maskinvare og programvare (4.3, 4.4), arbeidsfordeling (4.5) og dokumentasjon (4.6). Her beskrives altså verktøy og metoder som ble brukt for å komme frem til dagens løsning. Videre beskrives selve prosjektgjennomføringen (5), altså gjennomføringen av prosjektet. Her gjennomgås prosjektprosessen (5.1), man sammenstiller ønskede mål og oppnådde mål i måloppnåelse (5.2), ser på forventet timebruk mot reelt timebruk (5.3), før refleksjon rundt prosess (5.4) og til slutt endringer som avviker fra opprinnelig forstudie (5.5). Siste kapittel omfatter videre arbeid (6).

3 Oppgavebeskrivelse

Statens vegvesen ønsket et 'Proof of Concept' i form av en produksjonspipeline for maskinlæringsmodeller i GCP. Denne skulle kunne ta i mot en ferdig definert maskinlæringsmodell, og behandle denne slik at man på andre siden kunne hente ut prediksjoner gjennom API-kall. Målene definert i samarbeid med oppdragsgiver var at pipelinen skulle kunne ta i mot data, analysere og forberede data, samt dele data opp i trenings- og testsett som skulle brukes til å trene og teste modellens prediksjoner. Øvrige krav gikk ut på at man skulle kunne evaluere modellens prediksjonkvalitet, validere at den kan tas i bruk, levere modellen til et endepunkt og overvåke den når den er i bruk. Disse stadiene gjenspeiler løst prinsippene i MLOps. Prinsippene i MLOps er gjennomgått i designrapporten.[9] Oppgaven ble etter ønske fra arbeidsgiver løst med bruk av skyplattformen Google Cloud Platform.



Figur 2: Enkel oversikt over oppgaven.

4 Hvordan ble oppgaven løst

4.1 Litteratur og Internett

Informasjon som ble brukt til å løse oppgaven kommer først og fremst fra offisiell dokumentasjon på Internett. Helt konkret fra dokumentasjonssidene til GCP,[3] Tensorflow,[4] Kubeflow[5] og Kubernetes.[6] Det har også blitt utført nettsøk for øvrig informasjon i startfasen av prosjektet, spesielt da gruppen måtte lære seg opp i hva maskinlæring er og hvordan dette fungerer. Her ble blant annet nettsiden `https://medium.com` tatt flittig i bruk, da denne er kilde for mange artikler om emnet. I et konkret tilfelle valgte også gruppen å ta i bruk nettsiden Stack Overflow (`https://stackoverflow.com`) for å innhente hjelp, men dette førte ikke med seg noen løsning. Fysiske kilder ble ikke tatt i bruk.

4.2 Metoder og standarder

Prosjektdeltakerne har benyttet innslag fra utviklingsmetodene Scrum og Kanban i prosjektgjennomføringen, da især stand-ups hentet fra Scrum, samt Kanbantavler for oversikt over forefallende arbeid, hentet fra Kanban.

En stand-up er et kjapt møte, hvor prosjektdeltakere melder om hva de har arbeidet med siden sist og om det er noen utfordringer de ser med videre arbeid. Stand-upene ble gjennomført for å bidra til problemløsning, løse opp eventuelle blokkeringer som gjorde at folk stod fast, og for å oppdatere hverandre på fremdrift. Her har også veileder fra SVV, Torgeir Thoresen vært med og bidratt stort til problemløsing. Stand-ups har blitt gjennomført to ganger i uken gjennom prosjektperioden, henholdsvis tirsdager og torsdager,

med unntak av få dager, hvor stand-up har utgått grunnet ulike årsaker, som at prosjektgruppen har fokusert på andre emner i perioder.

Videre har prosjektgruppen, som nevnt, benyttet Kanbantavle for oversikt over forefallende arbeid. Til dette har verktøyet Trello (`https://trello.com/`) blitt brukt. Ved bruk av denne metoden har prosjektdeltakerne hatt god oversikt over hva som må gjøres i de ulike rapportene og utviklingsstegene, samt hvem som til enhver tid gjør hva.

Standarder har vært lite relevant for prosjektgruppen, men det har likevel blitt tatt utgangspunkt i etablerte maskinlæringsoperasjoner (MLOps) for å elevere prosessen fra en maskinlæringsmodell er utviklet til den ligger i produksjon. Dette innebærer standardprosedyrer en maskinlæringsmodell må gjennom før den er klar for å brukes. I dette prosjektets tilfelle har det vært aktuelt å trekke inn uthenting av data, analysering av data, forbereding av data, trening av modellen, evaluering av modellen, validering av modellen, levering av modellen og overvåkning av modellen. Grundigere beskrivelse av prosjektets bruk av MLOps kan leses i driftsrapporten.[10]

Gruppen har også forholdt seg til TensorFlow, som er de facto standard for maskinlæringsmodeller; ihvertfall for maskinlæring som baserer seg på nevrale nettverk.

I tillegg til dette har prosjektgruppen fulgt standarden IEEE som referansestil.

4.3 Maskinvare

Det ble ikke brukt noen maskinvare direkte, da løsningen ble utviklet og kjørt på Google Cloud Platform (GCP). Det var et ønske fra oppdragsgiver at dette skulle brukes fremfor frittstående maskiner. Metoden ved å utvikle en pipeline på en skyplattform fremfor lokale maskiner forenklet arbeidet på flere måter; bachelorgruppen slapp å forholde seg til fysiske maskiner direkte, ei heller installere tjenester på et underliggende operativsystem.

4.4 Programvare

Følgende programvare ble brukt under arbeidet med oppgaven:

Programvare	Brukes til
Google Cloud Platform	Skytjeneste som all underliggende programvare kjører på.
TensorFlow Extended (TFX)	Rammeverk for å definere pipeline.
Kubernetes	Plattform for å kjøre og skalere applikasjoner. Brukes for Kubeflow Pipelines som kjører pipelinen.
Kubeflow Pipelines (KFP)	Kjører selve pipelinen. Tar imot en kompilert pipeline fra TFX, og sørger for at komponentene kjøres i riktig rekkefølge.
Jupyter Notebooks	Brukes som utviklingsmiljø for pipelinen. Her kan TFX-kode utvikles, kompileres og sendes til KFP. Bachelorgruppen har brukt dette under utvikling, men dette er ikke nødvendig for videre utvikling.
AI Platform	Plattform for tjenester rund kunstig intelligens (KI) på GCP. AI Platform: Jobs brukes for trening av maskinlæringsmodeller, og pipelinen sender ferdige modeller til AI Platform: Models.

4.5 Arbeidsfordeling

Bachelorgruppen har hatt en viss arbeidsfordeling. Jon har hatt hovedansvar for utvikling, Erik har hatt hovedansvar for infrastruktur og vært formell prosjektleder, og Ingvild har hatt hovedansvar for dokumentasjon.

Alle tre bidro likt i starten med research. Etter hvert som arbeidet med oppgaven ble mer praktisk, falt arbeidsfordelingen naturlig ut i fra hvilke områder de forskjellige gruppemedlemmene har mest erfaring med.

Forutenom research, har Ingvild satt opp strukturen i alle rapporter, og skrevet nesten alt innhold som ikke er rent teknisk. Dette gjelder for designrapporten[9] og driftsrapporten.[10] Forstudierapporten[8] og sluttrapporten har hatt jevnere

fordeling av skriving.

Innledningsvis i utviklingen av pipelinen, jobbet Jon og Erik tett. Undersøkelse av hvilke tjenester som skulle brukes, og diverse feilsøking ble gjort sammen. Etter hvert som brikkene falt på plass, overtok Jon utviklingen av selve pipelinen, mens Erik fokuserte på infrastruktur. Avslutningsvis i implementasjon av løsning, jobbet Jon og Erik tilnærmet uavhengig med hvert sitt. Gruppemedlemmene oppdaterte likevel hverandre underveis gjennom stand-up-møter og jevnlig kommunikasjon ellers. Dette ble gjort slik at noen andre kunne steppe inn om den ansvarlige skulle bli syk eller lignende.

4.6 Dokumentasjon

Prosjektprosessen har ført til følgende dokumentasjon:

- Forstudierapport
- Designrapport
- Driftsrapport
- Readme
- Sluttrapport
- Prosjekthåndbok
- Presentasjon

Rapportene leveres skjøtet sammen i ett dokument. I tillegg blir alle vedlegg levert som en zip-fil. De ordinære vedleggene ligger under mappen *vedlegg*. Mappen *rapporter* inneholder de fire rapportene som enslige pdf-filer. Strukturen i zip-filen er slik:

```
vedlegg.zip
├── vedlegg/
│   ├── presentasjon.pptx
│   ├── readme.pdf
│   ├── prosjekthåndbok.pdf
│   ├── ordbok.pdf
│   └── kildekode.zip
└── rapporter/
    ├── forstudierapport.pdf
    ├── designrapport.pdf
    ├── driftsrapport.pdf
    └── sluttrapport.pdf
```

Figur 3: Struktur i zip-filen oppgaven leveres som

5 Gjennomføring av prosjektet

5.1 Prosess

Prosjektet ble delt inn i tre faser: oppstart, utførelse og ferdigstilling. Vanligvis vil utførelsesfasen være den desidert største fasen av et prosjekt, men i dette tilfellet var det mye nytt å sette seg inn i, så det ble brukt mye tid i oppstartsfasen. Under kommer tre lister med fasenes deloppgaver.

5.1.1 Oppstart

Arbeidet i denne fasen gikk til:

1. Planlegging med bedrift
2. Innføring i skymiljø
3. Utarbeide forstudierapport
4. Undersøke mulige teknologier og fremgangsmåter

Planleggingen med SVV startet allerede høstsemesteret 2020. Gruppen ble foreslått to forskjellige oppgaver, men valgte den vanskeligste, selv om SVV informerte om at det var en stor oppgave.

Etter valg av oppgave, kommuniserte ikke partene før vårsemesteret når

arbeidet skulle starte. Det ble klargjort grundigere hva SVV var ute etter. Sammen med SVV kom partene frem til en rekke behov. Oppgaven ble konkret å utarbeide og/eller undersøke hvordan man kunne implementere funksjonalitet som støttet opp under disse behovene. Det ble foretatt et møte for å utarbeide brukerhistorier for pipelinen.

I januar fikk bachelorgruppen en innføring i GCP av veileder Torgeir Thoresen. En oversikt over forskjellige funksjoner i GCP, samt forklaring av miljøer, tjenestebrukere og andre ting som kunne skape forvirring om man ikke var informert om deres funksjon. Samtidig ble det gjort avtaler på at det skulle avholdes møter annenhver uke for å oppdatere prosjekteier Lars Meisingseth, sensor fra SVV Safurudin Mahic og veileder fra NTNU, Jostein Lund, med status underveis. Dette ble kun gjennomført et par ganger, da det viste seg lite hensiktsmessig med disse møtene i dette prosjektet. Stand-ups ble dekkende nok for prosjektgruppen og oppdragsgiver.

Arbeidet med å skrive forstudierapport startet helt i begynnelsen av semesteret, og foregikk parallelt med de to tidligere punktene. Selve arbeidet med forstudierapporten gikk stort sett knirkefritt, da bachelorgruppen hadde god erfaring med å utarbeide slike rapporter fra tidligere i studiet. Et par kapitler, spesielt de som omhandlet mål og brukerhistorier, krevde ekstra kommunikasjon med SVV før de kunne skrives.

Den siste delen av oppstartsfasen, som omhandler undersøkelse av teknologier og fremgangsmåter var spesielt tidkrevende i dette prosjektet. Denne fasen ble brukt som utgangspunkt for resten av arbeidet i oppgaven, og måtte bli gjort på en god måte for at gruppen i det hele tatt skulle kunne klare å levere på noen mål i det hele tatt.

Det ble lest mange medium-artikler og fulgt mange Jupyter-notebooker. De førstnevnte var stort sett veldig vage, og de sistnevnte fungerte ikke. Gruppen forstod at dette var et veldig nytt felt som var lite dokumentert. Gruppen hadde blitt varslet om dette fra SVV allerede første møte.

Etter noen ukers graving, oppdaget gruppen TensorFlow Extended (TFX). Tilsynelatende kunne dette rammeverket brukes for å løse de oppgavene gruppen hadde fått. En veldig lovende Youtube-video[2] som ga et overblikk

over rammeverket overbeviste gruppen om at det var dette som skulle brukes. De innebygde komponentene i TFX kunne brukes uten videre konfigurasjon. Etter ønske var det også mulighet til å tilpasse de innebygde komponentene eller skrive helt nye komponenter. Dette i tillegg til å kunne bruke hvilken som helst orkestrator var de to viktigste punktene som gjorde at gruppen valgte dette rammeverket. Gruppen var også frustrerte av å tilsynelatende ikke komme noen vei med prosjektet etter å ha undersøkt i flere uker, og var veldig klare for å komme seg videre i prosjektet.

Det var tilgjengelig Jupyter-notebooker for å sette opp grunnleggende TFX-pipeliner, så gruppen valgte å ta utgangspunkt i dette for å bruke rammeverket. Dessverre fungerte ikke disse, og gruppen forstod at det krevde bedre dybdekunnskap for å skulle få til prosjektet. Jon brukte om lag to uker på å gjennomgå hver enkel TFX-komponent for å forstå hver enkelt komponents rolle i pipelinen, og hvordan disse skulle konfigureres. Mens Jon satte seg inn i TFX, testet Erik de individuelle funksjonene i GCP som ville bli brukt senere av pipelinen for å integrere pipelinen med GCP.

En prototype ble utarbeidet etter den dypere undersøkelsen av TFX. En populær orkestrator som virket lovende var Kubeflow Pipelines (KFP). En orkestrator krevdes for å kjøre pipelinen, men gruppen hadde litt vanskeligheter med å få brukt KFP helt i starten. Etter en vurdering av alternativer, fant gruppen ut at KFP sannsynligvis var det rette valget, og at det ville være verdt arbeidet med å finne ut hvordan man satt opp dette. Erik tok ansvar for dette, og til slutt hadde gruppen en teknisk sett fungerende pipeline.

En gjennomgående utfordring under hele prosjektet var at MLOps er et veldig nytt felt, som fører til at utarbeidelse av pipeliner og feilsøking av disse er dårlig dokumentert. Til dels svak eller manglende dokumentasjon på TFX, samt fraværet av diskusjoner på Stack Overflow, gjorde at gruppen måtte klare seg foruten krykker som en ellers er vant til å ha i andre utviklingsprosjekter.

5.1.2 Utførelse

Arbeidet i denne fasen gikk til å:

1. Utarbeide designrapport
2. Implementere tiltenkt løsning
3. Utarbeide driftsrapport

Å si at utførelsesfasen kom etter oppstartsfasen er ikke helt korrekt. Arbeidet med å skrive designrapport startet samtidig som gruppen undersøkte teknologier. Dette er naturlig, da designrapporten skal rapportere det en har undersøkt og bestemt seg for.

Selve arbeidet med designrapporten gikk fint uten betydelige utfordringer. Etter gruppen hadde landet på en løsning og valg av teknologi, handlet det bare om å få nedfelt dette i rapporten.

Neste steg var å implementere selve løsningen som var skissert i designrapporten. Dette var utfordrende, men med bakgrunn i forarbeidet som ble gjort i forrige fase, hadde gruppen gode forutsetninger for å få gjennomført oppgaven. Ved hjelp av en eksempelpipeline som kan finnes som mal i TFX, kunne gruppen utvikle en hel, fungerende pipeline. De elementer som ble brukt fra malen var konfigurasjon som var spesifikk for en bestemt modell og data. Resten av arbeidet, som gikk på å sette sammen pipelinen med denne konfigurasjonen var altså helt generell. Konfigurasjonsfilene fra malen ble brukt for å teste pipelinen. Tilsvarende konfigurasjonsfiler skulle SVV utvikle på egenhånd i ettertid, og var utenfor bacheloroppgavens rammer.

Det oppstod stadig feilmeldinger for hver nye komponent som ble lagt til pipelinen. Stort sett var det vanskelig å tyde feilmeldingene, og som tidligere nevnt var det mangel på dokumentasjon og foruminnlegg om disse problemene. Problemene ble stort sett løst gjennom prøving og feiling. Stort sett var det den spesifikke konfigurasjonen for de enkelte komponentene som skapte trøbbel. En ting gruppen ikke fikk rettet på, var mellomlagring av utdata. Dette viste seg å skyldes en bug i KFP, så mellomlagring måtte skrus av for at pipelinen skulle kjøre.

Til slutt hadde gruppen en fullt fungerende pipeline som kjørte i KFP. Neste steg var å integrere de individuelle stegene i pipelinen med funksjoner i GCP. Dette var en forholdsvis enkel sak, og gruppen var takknemlig for at de hadde

valgt TFX, da dette er en av rammeverkets sterke sider.

Etter gruppen hadde utviklet en pipeline som kjørte helt og holdent på GCP med GCP-funksjoner i hvert ledd, gikk resten av arbeidet ut på å legge til rette for videre utvikling. Koden som var utviklet gjennom en lang prosess med prøving og feiling, var som et resultat blitt meget rotete. Rydding i dette, flytting og samling av konfigurasjon og kommentering av kode preget resten av selve utviklingen av pipelinen. Jon tok ansvar for å ferdigstille pipelinen, mens Erik lagde skript for å automatisk sette opp infrastrukturen som krevdes av pipelinen på GCP. I denne perioden vekslet Ingvild på å bistå Jon og Erik og på å skrive driftsrapport.

Bachelorgruppen hadde tidligere i prosjektet etterspurt en maskinlæringsmodell fra Statens vegvesen for å teste pipelinen med denne. Maskinlæringsteamet til Statens vegvesen var ikke klar med en modell som kunne brukes til testing før i slutten av april. På dette tidspunktet var bachelorgruppen opptatt med å runde av utviklingen av pipelinen og dokumentere denne, så dette var dessverre litt for sent. Maskinlæringsteamet til Statens vegvesen virket positive til pipelinen som var blitt utviklet. De satt av to dager til å sette seg inn i hvordan pipelinen kunne tilpasses modellen sin. Mye tid gikk til å få kjørt pipelinen lokalt, fremfor på GCP, for å forenkle utviklingen. Videre oppstod det problemer med å skulle gjøre all preprosessering av data bare ved bruk av TensorFlow-tensorer.

Videre skulle driftsrapporten skrives. Ingvild satte i gang arbeidet med dette samtidig som Jon og Erik gjorde det avsluttende praktiske arbeidet. Selve arbeidet med driftsrapporten gikk nok en gang greit, om noe tidkrevende. Her var det mye teknisk dokumentasjon som skulle skrives. Samtidig som Jon og Ingvild jobbet med driftsrapporten, skrev Erik en readme[12] som forklarer hvordan infrastrukturen og pipelinen skulle settes opp på GCP. Erik skrev også på driftsrapporten om infrastruktur.

Den største fartshumpen som bachelorgruppen støtte på kom helt på slutten, da de måtte ta en og en halv uke pause fra jobbingen med oppgaven for å lese til eksamen i IDRI2006 Organisasjon og ledelse. Etter dette hadde gruppen en drøy uke på å skrive ferdig driftsapporten og sluttrapporten.

5.1.3 Ferdigstilling

Arbeidet i denne fasen gikk til å:

1. Utarbeide sluttrapport
2. Levere oppgave
3. Presentere oppgave

Arbeidet med sluttrapporten gikk greitt for seg. Her handlet det bare om i løpet av en uke få oppsummert hele prosjektet i en rapport. Etter en jevn fordeling av innhold som skulle skrives av hver enkelt i gruppen, har arbeidet med sluttrapporten vært et av de enklere stegene i prosjektet. Gruppen hadde så god tid at de tok seg råd til fri på 17. mai.

Oppgaven leveres 20. mai. Selve leveransen av oppgaven vil inneholde denne rapporten og alle andre rapporter tidligere skrevet i prosjektet. Den vil også inneholde ordbok, prosjekthåndbok, readme, kildekode og lysbildene for presentasjonen. Til slutt vil leveransen inneholde en zip av 11. mai-utgaven av GitHub-pakkebrønnen.[7] Alle disse tingene leveres samlet i en enkelt zip-fil.

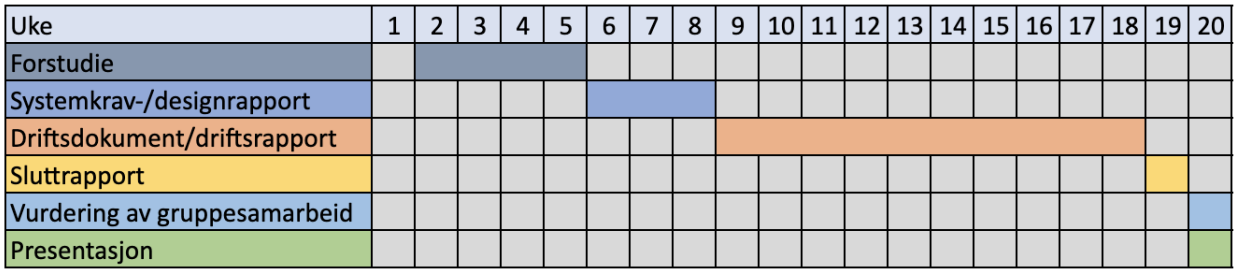
Gruppen har avtalt tidspunkt for presentasjon av oppgaven. Dette skjer 21. mai klokken 15:00. Invitert er veileder fra NTNU, Jostein Lund, veileder fra Statens vegvesen, Torgeir Thoresen, sensor fra Statens vegvesen, Safurudin Mahic, produkteier hos Statens vegvesen, Lars Meisingseth og emneansvarlig hos NTNU, Stein Meisingseth.

Statens vegvesen har også etterspurt en ekstra presentasjon som heller vil ha fokus på prosjektets produkt og vil bli en teknisk gjennomgang for ansatte og en innføring i hvordan dette kan brukes i praksis. Denne faller den 25. mai klokken 13:00.

5.1.4 Planlagt og faktisk arbeidsprosess

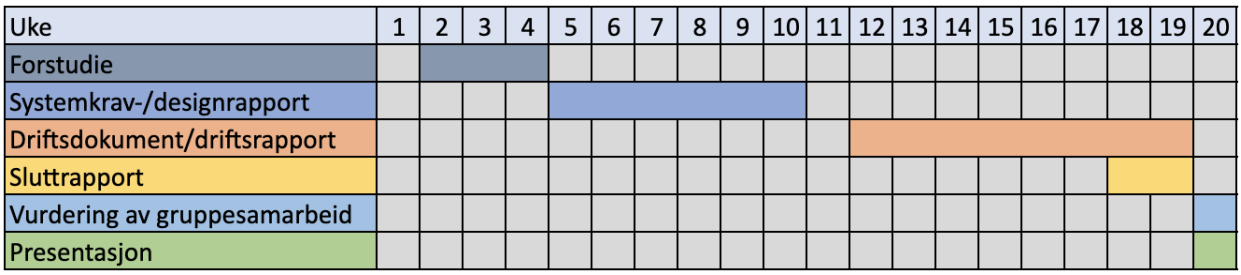
Den planlagte arbeidsprosessen er hentet fra antatt tidsbruk per rapport, på bakgrunn av tidligere bacheloroppgaver fra instituttet. Det har vist seg vanlig å bruke 2-5 uker på forstudierapport, avhengig av oppgavens omfang. Prosjektgruppen satte derfor opp fire uker til dette, da forhåndskunnskapene rundt teknologi og løsning var relativt liten. Videre har designrapporten et

gjennomsnittlig tidsomfang på 2-3 uker. Driftsrapport er den rapporten som vanligvis opptar mest tid, 8-12 uker. Prosjektgruppen budsjetterte derfor med 10 uker til denne, for at tiden skulle brukes opp. Til slutt skal sluttrapport, vurdering av samarbeid og presentasjon være relativt lite tidkrevende. Se figur 4 for prosjektgruppens stipulerte arbeidsprosess.



Figur 4: Stipulert arbeidsprosess

Under følger prosjektgruppens faktiske arbeidsprosess (figur 5).



Figur 5: Faktisk arbeidsprosess

Av dette kan en se at det ble brukt en uke mindre enn planlagt på forstudie, tre uker mer på designrapport, to uker mindre på driftsrapport og en uke ekstra på sluttrapport, samt noe forflytninger på oppstart og slutt med rapportene. Rapportene ble utarbeidet parallelt med utforsking og testing av teknologi underveis, hvor uke 11 gikk kun til testing av valgt teknologi. For grundigere Gantt-diagram, se underkapittel **??** Gantt-diagram.

5.2 Måloppnåelse

Målene som ble satt for prosjektet, kan finnes igjen i kapittel 3 i forstudierapporten.[8]

5.2.1 Effektmål

Effektmålene som ble satt, er som følger:

- *Ta i bruk mer moderne teknologi i veisektoren.*

  Skulle Vegvesenet ta i bruk pipelinen som bachelorgruppen har utviklet, vil dette målet være oppnådd.

- *Innhente forslag til hvordan de kan kjøre det foreslåtte systemet i produksjon.*

  Dette er essensen i bacheloroppgaven. Pipelinen som bachelorgruppen har levert, er både systemet og i seg selv et eksempel på hvordan man kan kjøre dette i produksjon.

- *Et konkret produkt som kan testes ut.*

  Pipelinen er et konkret produkt som kan testes ut.

- *Kunne vurdere om dette er retningen de ønsker å bevege seg i, eller om de vil utforske alternativer.*

  Oppgaven er et forslag til hvordan man kan kjøre maskinlæringspipeliner i produksjon, og danner et beslutningsgrunnlag for nettopp dette målet.

5.2.2 Resultatmål

Resultatmålene som ble satt, er som følger:

Hovedmål:

- *Et ferdigstilt 'Proof of Concept' på en produksjonspipeline for en maskinlæringsmodell i GCP.*

  Dette har bachelorgruppen utviklet.

- *Ha resultatet klart innen 20.05.2021*

  Resultatet er klart, og levert innen 20.05.2021.

Delmål:

- Kunne kjøre ut en maskinlæringsmodell i GCP.
- Kunne kjøre ut nye versjoner av en maskinlæringsmodell i GCP.

- Kunne eksponere en maskinlæringsmodell for input og bruke den fra andre applikasjoner.
- Finne ut hvordan man kan overvåke en maskinlæringsmodell og dens evne til å predikere riktig.
- Finne ut hvordan man kan skalere en pipeline med hensyn på kapasitet og ytelse.

Alle disse delmålene er nådd. En grundig forklaring for hvordan og hvorfor hvert mål har blitt nådd, finnes i kapittel 6 i driftsrapporten.[10]

Det ble tatt forbehold om at gruppen ikke nødvendigvis ville klare å nå alle delmålene, og kunne snevre inn oppgaven om det ville være behov for dette. Derimot var ikke dette nødvendig, og bachelorgruppen har klart å levere på alle mål.

5.2.3 Prosessmål

Prosessmålene som ble satt, er som følger:

- *Økt kunnskap innen GCP og dens underliggende verktøy*

  Gruppen har absolutt blitt bedre kjent med GCP. Ingen på gruppen hadde noe erfaring med GCP ved prosjektstart.

- *Kompetanse på hvordan man planlegger og innfører en produksjonspipeline for maskinlæringsmodeller i GCP*

  Gruppen har lært hvordan man designer og innfører en produksjonspipeline for maskinlæringsmodeller kun ved bruk av tjenester på GCP.

- *Styrkede samarbeidsevner i teamarbeid med færre gruppemedlemmer*

  Gruppen hadde allerede god erfaring med å samarbeide med hverandre, og fikk videreutvikle dette. Samarbeidet fungerte godt, selv om medlemmene var vant med å være litt flere på en gruppe.

- *Økt kunnskap om individuelle styrker og svakheter i gruppesammenheng*

Gruppemedlemmene har blitt bedre kjent med hverandres styrker og svakheter når det kommer til samarbeid og kompetanse.

- *Erfaring rundt kommunikasjon og samarbeid med en ekstern, reell bedrift*

  Dette var en ny erfaring for gruppemedlemmene, og alle medlemmer har definitivt fått mye mer erfaring med dette enn de hadde i begynnelsen.

- *Høy faglig måloppnåelse*

  Dette gjenstår å se, men gruppen er optimistisk.

5.3 Timeregnskap

Se Gantt-diagram, ukesrapport og timefordeling i vedlagt prosjekthåndbok.[13]

5.4 Refleksjon rundt prosessen

Det positive med oppgaven var at den ga bachelorgruppen muligheten til å tilegne seg mye ny kunnskap innen maskinlæring, MLOps og Google Cloud Platform. Listen over krav fra Statens vegvesen var fleksibel, noe som la opp til at gruppen kunne konsentrere seg om å skape en best mulig løsning fremfor å fokusere på å få med alt". Det var både spennende og utfordrende å sette seg inn i et såpass lite utprøvd felt, noe som ga både mestringsfølelse og frustrasjon om hverandre. Oppgavens omfang ble konkretisert gjennom ukentlige møter med Statens vegvesen, slik at løsningen aldri lå i fare for å havne utenfor hva oppgavestiller ønsket. Denne tette oppfølgingen var svært positiv, da det ga en trygghet for både bachelorgruppen og Statens vegvesen. Målene som ble satt i forstudien ble nådd med god margin, selv om man til tider fryktet at noe funksjonalitet måtte sløyfes. Gruppen sitter igjen med mye nyttig kunnskap om Google Cloud Platform og drifting av maskinlæringsmodeller, samt hvordan det er å arbeide med en slik oppgave tett opp mot en bedrift. Tidsrammene ble litt flytende da oppgaven behøvde mye tid på research, men fungerte godt sett i sin helhet. Dette ble løst ved at man delegerte ut arbeidsansvar, slik at man kunne jobbe i parallell. Denne metoden fungerte tilfredsstillende for hele prosjektgruppen. Samarbeid mellom prosjektgruppen, bedrift og veileder har i hovedsak skjedd over

Microsoft Teams og Slack. Førstnevnte fungerte godt innen det mer formelle samarbeidet, der bedrift og veileder hadde tilgang på dokumenter, samt en oversikt over felles møter i kalender. Slack var en mer uformell plattform mellom bachelorgruppen og bedrift, der man kunne stille spørsmål om oppgaven når man sto fast. Denne fungerte også godt, og ga gruppen muligheten til å be raskt om innspill utenfor møtetidene. Det kan også trekkes frem at oppgaven er nyttig for bedriften, noe som gir en mening rundt utførelsen.

Det er mye man i ettertid kan si at skulle ha blitt gjort annerledes. Hadde man skrevet en slik refleksjon i starten av prosessen kunne det blitt argumentert for at oppgaven var for krevende, da gruppen sto mye fast. Dette løste seg fint, noe som heller ga en erfaring om at hvis man jobber på så går det bra. I gruppens interne prosess kan man kritisere at det ble brukt mye tid på research i startfasen som viste seg å ha lite funksjon for sluttproduktet, og at man skulle ha begrenset omfanget av teknologi tidligere. Spesielt ble man sittende mye med gjennomganger fra offisiell dokumentasjon som ikke fungerte. Som nevnt under prosesskapittelet måtte gruppen til slutt bruke mye tid på å gå igjennom hver enkelt komponent, noe man kunne ha satt i gang med tidligere. Bachelorgruppen må også ta selvkritikk for ikke å bruke tid på å sette seg inn i kostnadsfunksjonene i Google Cloud Platform, da det i første del av prosessen ble høye kostnader rundt utforskingen inne i GCP. Når det kommer til dokumentasjon valgte gruppen å bruke Overleaf/LaTeX fremfor den mer tradisjonelle Microsoft Word. Da størsteparten av medlemmene var mer vant med Word i begynnelsen, tok dette noe ekstra tid i startfasen av prosjektet, selv om det fungerte godt når alle lærte teknologien.

Et negativt aspekt ved perioden som må trekkes frem er Covid-situasjonen, som førte til at gruppen ikke kunne arbeide fysisk hos Statens vegvesen som planlagt. Dette kunne påvirket oppgaven i en dårlig retning, da effektiv kommunikasjon er mye vanskeligere over nett. Veileder hos Statens vegvesen tilpasset seg godt til situasjonen, og hans tilstedeværelse to ganger i uka gjennom hele perioden førte til at konsekvensen av Covid ikke var så stor som den kunne ha blitt. Intern veileder hos NTNU var også tilgjengelig på videosamtale og e-post når dette var nødvendig. Covid skapte også problemer internt i gruppa. Da dette var en praktisk oppgave fungerte gruppa

best under fysisk samarbeid, og var derfor avhengig av å finne grupperom på NTNU. Dette fungerte dårlig i de periodene NTNU stengte campus, i tillegg til at det var vanskelig å booke grupperom i tide. I en ideell situasjon hadde gruppa hatt tilgang til et fast arbeidssted gjennom hele perioden.

5.4.1 Suksessfaktorer og risikovurdering

I begynnelsen av prosjektperioden ble det satt opp et sett av suksessfaktorer. I en refleksjon rundt prosessen kan det være nyttig å se tilbake på disse. For det første ble det sett på som nødvendig med aktiv involvering og tilrettelegging fra Statens vegvesen. Som nevnt tidligere i refleksjonen stilte ekstern veileder opp over all forventning, og alt gruppen behøvde hjelp med ble tilrettelagt. Disse hyppige møtene oppfylte også suksessfaktoren som handlet om regelmessig informasjonsutveksling. De neste suksessfaktorene baserte seg på tilgjengelighet i Google Cloud Platform og at løsningen måtte basere seg på GCP-komponenter. Gruppen har ikke opplevd noen problemer med funksjonaliteten til GCP, og klarte å utvikle en løsning som kjører på skyplattformen. All funksjonalitet ble godt dokumentert, og etter mye prøving og feiling klarte gruppen å tilegne seg tilstrekkelig kunnskap til å utvikle ønsket løsning.

En annen analyse som ble gjort i starten av prosessen er risikoanalysen. De uønskede hendelsene som ble beskrevet her er også relevante å trekke frem i en diskusjon om prosessen. Som nevnt tidligere i refleksjonen var gruppen i begynnelsen bekymret for at oppgaven hadde for stort omfang, og ga denne hendelsen moderat sannsynlighet. Dette viste seg å være feil, og gruppen klarte å møte de kravene som ble satt sammen med oppgavestiller. Andre risikofaktorer som ble trukket frem var sykdom og konflikt i gruppa. Gruppa har samarbeidet godt, og har ikke vært rammet av verken Covid eller øvrig sykdom. Videre risiko omhandler problemer med selve teknologien, spesielt at produktet ikke fungerer til demo. Når arbeidet løsnet etter en lang periode med research, fungerte teknologien fint. Det var mye feilsøkning, men det er noe man kan forvente i et slikt prosjekt. Gruppen har allerede holdt en demo for Statens vegvesen og intern sensor som har vært vellykket, og forventer derfor det samme på offisiell presentasjon av oppgaven.

5.5 Endringer

Endringer omfatter endringer i løsning og teknologi.

Arbeidet med oppgaven har vært såpass vellykket at det ikke har vært behov for endring. Å bruke ekstra lang tid på å orientere seg rundt teknologier lønnet seg, da valget av teknologi viste seg å fungere godt. Det har ikke blitt foretatt noen utskiftninger i teknologistakken.

Det ble i begynnelsen av prosjektet tatt forbehold om at det kunne være behov for å innsnevre oppgaven om det skulle være behov for dette. Gruppen har klart å nå alle målene som ble satt opprinnelig. En liten avklaring ble gjort, nemlig at pipelinen kun vil fungere med Tensorflow-modeller.

6 Videre arbeid

Det neste som gjenstår med pipelinen, er å sette denne i produksjon. Her må maskinlæringsteamet hos Statens vegvesen legge inn en innsats for at pipelinen skal passe til den spesifikke maskinlæringsmodellen de utvikler. Dette faller utenfor omfanget til prosjektet, men dokumentasjonen som gruppen har skrevet i driftsrapporten[10] vil være til hjelp med dette arbeidet.

En grundigere gjennomgang om videre arbeid og potensielle forbedringer av produktet, finnes i kapittel 5,4 i driftsrapporten.[10]

Referanser

[1] Regjeringen: *Statens vegvesen*, 2021.
Hentet fra:
`https://www.regjeringen.no/no/dep/sd/org/underliggende-etater/statens-vegvesen/id443412/`.
Lastet ned: 15. januar 2021

[2] Youtube: *TFX: Production ML pipelines with TensorFlow (TF World '19)*, 2019.
Hentet fra:
`https://www.youtube.com/watch?v=TA5kbFgeUlk`.
Lastet ned: 12. mai 2021

[3] Google: *Documentation | Google Cloud*, 2021.
Hentet fra:
`https://cloud.google.com/docs/`.
Lastet ned: 20. april 2021

[4] Tensorflow: *TFX API Reference*, 2021.
Hentet fra:
`https://www.tensorflow.org/tfx/api_overview/`.
Lastet ned: 20. april 2021

[5] Kubeflow: *Kubeflow Pipelines*, 2021.
Hentet fra:
`https://www.kubeflow.org/docs/components/pipelines/`.
Lastet ned: 20. april 2021

[6] Kubernetes: *Kubernetes Documentation*, 2021.
Hentet fra:
`https://kubernetes.io/docs/home/`.
Lastet ned: 20. april 2021

[7] GitHub: *Bacheloroppgave for Statens vegvesen - Release Innlevering av bachelorarbeid*, 2021.
Hentet fra:
`https://github.com/efdalen/bachelor2021/releases/tag/1`.
Lastet ned: 11. mai 2021

[8] J. A. Langholm, I. Andersen og E. F. Dalen, ”Forstudierapport”, *NTNU*, Februar 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[9] J. A. Langholm, I. Andersen og E. F. Dalen, ”Designrapport”, *NTNU*, Mars 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[10] J. A. Langholm, I. Andersen og E. F. Dalen, ”Driftsrapport”, *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[11] J. A. Langholm, I. Andersen og E. F. Dalen, ”Ordbok”, *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[12] J. A. Langholm, I. Andersen og E. F. Dalen, ”README”, *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

[13] J. A. Langholm, I. Andersen og E. F. Dalen, ”Prosjekthåndbok”, *NTNU*, Mai 2021. [Vedlegg].
Hentet fra:
Vedlegg i innlevering av oppgave.

I. Andersen, J. A. Langholm og E. F. Dalen

MLOps på Google Cloud Platform

NTNU
Kunnskap for en bedre verden