

# Tekoälyavusteisen ja perinteisen ohjelmistoprojektin erot

Sami Saastamoinen

OPINNÄYTETYÖ  
Kesäkuu 2024

Tietotekniikan tutkinto-ohjelma  
Ohjelmistotekniikka

## TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tietotekniikan tutkinto-ohjelma  
Ohjelmistotekniikka

SAASTAMOINEN, SAMI:

Tekoälyavusteisen ja perinteisen ohjelmistoprojektin erot

Opinnäytetyö 41 sivua  
Kesäkuu 2024

---

Opinnäytetyössä toteutettiin kaksi ohjelmistoprojektia, perinteiseksi määritelty sekä tekoälyavusteinen. Projektien valmistuttua niiden toteutustavoissa ilmentyneitä yhtäläisyyksiä ja eroavaisuuksia vertaillaan toisiinsa. Työn perimmäisenä tarkoituksena oli tarkastella, voitaisiinko tekoälytyökalujen hyödyntämisellä mahdollisesti helpottaa ohjelmistokehittäjien työtaakkaa.

Projektien lisäksi opinnäytetyössä käsiteltiin tekoälyn historiaa sekä tekoälyn erilaisia määritelmiä ja teknologioita. Työssä käytiin läpi termin alle kuuluvia osaluokkia. Työssä tutustutaan myös joihinkin tekoälyn sovelluksiin.

Työssä toteutettavat projektit rajattiin käyttämään Next.js-sovelluskehystä sekä Tailwind CSS -tyylittelykehystä. Muut projekteissa tarvittavat kirjastot valittiin projektikohtaisesti toteutusvaiheessa. Projektien aiheena oli toteuttaa datan visualisointia hyödyntävä web-työkalu.

Opinnäytetyössä suoritettujen projektivertailun tuloksena huomattiin hyötyjä toteutusnopeudessa sekä -mieluisuudessa tekoälytyökalujen käytöstä. Vertailussa huomattiin myös, että tekoälyn vastaukset sisältävät usein virheitä tai muokkautarpeita. Tiedonhaun ja käytettävien teknologioiden dokumentaatioon perehtymisellä ei välttytty tekoälyn hyödyntämisestä huolimatta.

---

Asiasanat: ohjelmistoprojekti, ohjelmistokehitys, tekoäly, datavisualisointi

## ABSTRACT

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Degree Programme in ICT Engineering  
Software Engineering

SAASTAMOINEN, SAMI:

Differences between an AI-assisted and a traditional software project

Bachelor's thesis 41 pages

June 2024

---

The objective of this thesis was to implement two different software projects, one deemed as a traditional project and one using AI-powered development tools. After completing these projects, their project workflows, similarities and differences were compared. The purpose of these comparisons was to determine if using these AI tools could potentially be used to improve the workflow of software developers.

The beginning of this thesis includes theoretical chapters which cover the history of artificial intelligence and the different definitions of AI. What sub-branches and technologies fall under the term artificial intelligence is also considered. This part of the thesis also introduces some AI-based tools.

The software projects were defined to be using Next.js framework as the basis for the web tool. The styling of these tools was to be handled by Tailwind CSS - framework. Other necessary parts and libraries needed to complete the projects were decided for each project during the implementation process. The goal for these projects was to create a web tool that utilises data visualisation.

The findings of this study indicate that usage of AI tools in project development can increase the pace of development and make the experience more enjoyable. In the study it was also found that answers given by the AI tools could often contain mistakes or need further modifications. Consulting the documentation of the utilised technologies also cannot be replaced entirely by using these AI tools.

---

Key words: software project, software development, ai, artificial intelligence, data visualisation

## SISÄLLYS

1	JOHDANTO .....	6
2	TEKOÄLY .....	7
	2.1 Tekoälyn historia ja aikakaudet .....	7
	2.1.1 AI Winter .....	9
	2.2 Tekoälyn määritelmä .....	10
	2.2.1 Turingin testi .....	11
3	TEKOÄLYN TEKNOLOGIA .....	13
	3.1 Tekoälyn osa-alueet .....	13
	3.1.1 Koneoppiminen .....	14
	3.1.2 Luonnollisen kielen käsittely .....	16
	3.2 Suuret kielimallit .....	16
	3.3 Tekoälyn sovellukset .....	18
	3.3.1 ChatGPT .....	18
	3.3.2 GitHub Copilot .....	19
4	PROJEKTIEEN RAJAUS JA TEKNOLOGIAT .....	22
	4.1 Perinteinen ohjelmistoprojekti .....	23
	4.2 Tekoälyavusteinen ohjelmistoprojekti .....	23
	4.3 React .....	24
	4.3.1 Next.js .....	24
5	PROJEKTIEEN TOTEUTUS .....	26
	5.1 Perinteinen ohjelmistoprojekti .....	26
	5.2 Tekoälyavusteinen ohjelmistoprojekti .....	28
	5.3 Projektien yhteenveto .....	31
6	TULOSTEEN VERTAILU .....	33
	6.1 Projektien toteutusnopeus .....	33
	6.2 Kehitystyökalujen käyttäjäystävällisyys .....	34
	6.3 Virheenkäsittely .....	36
	6.4 Toteutusten virheellisyys .....	36
	6.5 Projekteista oppiminen .....	37
7	POHDINTA .....	38
	LÄHTEET .....	39

**LYHENTEET JA TERMIT**

AAAI	The Association for the Advancement of Artificial Intelligence (entinen the American Association of Artificial Intelligence)
AI	Artificial Intelligence (Tekoäly)
CV	Computer Vision
DSRPAI	Dartmouth Summer Research Project on Artificial Intelligence.
GPT	Generative pre-trained transformers
JavaScript	Ohjelmointikieli, jota käytetään pääasiassa web-soveluksissa.
JSX	JavaScript XML, Metan kehittämä syntaksilaajennus JavaScriptille, joka yhdistää siihen HTML-toiminnallisuutta.
ML	Machine Learning (Koneoppiminen)
NLP	Natural Language Processing (Luonnollisen kielen käsittely)
SPA	Single Page Application (Yhden sivun applikaatio) mahdollistaa web-sivun tai -applikaation osittaisen dynaamisen päivityksen.
SSO	Single sign-on (Kertakirjautuminen) mahdollistaa pääsyn useisiin palveluihin yhdellä todennuskerralla.
TypeScript	Ohjelmointikieli, joka laajentaa JavaScriptiä lisäämällä vahvan tyyppityksen sen syntaksiin.
Visual Studio Code	Microsoftin kehittämä koodieditori.

## 1 JOHDANTO

Nykyisin usein kuultu toteamus on, että tekoäly vie ohjelmistokehittäjien työpaikat. Mutta onko tilanne todella niin vakava? Ainakin generatiivisen tekoälyn sovellukset kehittyvät kovaa tahtia yhä nopeammiksi, tehokkaammiksi ja yleistietävämmiksi.

Tämän opinnäytetyön tavoitteena on selvittää, miten projektin toteuttaminen perinteiseksi määriteltävällä tavalla poikkeaa saman laajuisen ja sama-aiheisen projektin toteuttamisesta tekoälytyökalujen avustuksella. Työn projektiosuus koostuu kyseisistä kahdesta projektista, joiden aikana luodaan verkkopohjainen datavisualisoinnin työkalu. Projektien toteuduttua niiden lopputuloksia sekä toteuttamisessa ilmenneitä vaiheita, yhtäläisyyksiä ja eroja koostetaan sekä vertailaan keskenään.

Ennen projektiosuutta työssä käydään läpi tekoälyn historiaa, käsitellään tekoälyn erilaisia määritelmiä sekä perehdytään tekoäly-termin alle kuuluviin eri osaluaisiin. Työssä avataan myös hieman suurien kielimallien toimintaa ja tutustutaan tekoälyn sovelluksiin. Toteutettavat projektit myös rajataan työn yhteydessä.

Työstä saatavat lopputulokset pohjautuvat projektien omatoimiseen pohdiskelvaan vertailuun sekä omaan näkemykseen perustuvaan kritisointiin. Tämän takia tulokset voivat sisältää henkilökohtaiseen mieltymykseen pohjautuvia seikkoja projektin toteutustapoihin liittyen.

## 2 TEKOÄLY

### 2.1 Tekoälyn historia ja aikakaudet

Tekoälyn alkuaikajankohtaa on hankalaa tarkkaan määritellä, mutta konseptina tekoälyn sekä älykkäiden koneiden sanotaan saavan alkunsa 1940–1950-lukujen aikoihin. Vuonna 1942 tieteiskirjailija Isaac Asimov julkaisi robotiikan kolmea lakia käsittelevän novellinsa ”Runaround”, jonka katsotaan inspiroineen monia tietotekniikan, robotiikan sekä tekoälyn alojen tieteilijöitä (Lu 2022).

Seuraavana askeleena matkalla kohti nykypäivän tekoälyn pidetään matemaatikko Alan Turingin artikkelia ”Computing Machinery and Intelligence”. Turing alkoi pohtimaan koneiden älykkyyttä kehittäessään Bombe-salauksenpurkulaitetta, jolla pystyttäisiin purkamaan Enigma-koneen salaamia viestejä toisen maailmansodan aikaan. Artikkelissaan Turing kuvailee älykkäiden koneiden luomista sekä keinoja niiden älykkyyden testaamiseen. Tästä kyseisestä artikkelista on peräisin myös Turingin testi, joka vielä nykyäänkin on tunnettu menetelmä koneiden älykkyyden tunnistamisessa. (Haenlein & Kaplan 2019.) Näistä syistä Turingia pidetäänkin yhtenä teoreettisen tietojenkäsittelytieteen ja tekoälyn esikuvana ja isänä (Lu 2022).

Terminä tekoäly (Artificial Intelligence) otettiin virallisesti ensimmäistä kertaa käyttöön vuonna 1956 Dartmouthin yliopistossa suoritettuna tekoälyn tutkimusprojektin (DSRPAL) yhteydessä. Projektiin osallistuivat muun muassa Nathaniel Rochester, Claude Shannon, Marvin Minsky sekä John McCarthy. (Haenlein & Kaplan 2019.) Tämä tutkimus sekä sitä edeltävät edistymiset tekoälyn saralla muodostavat tekoälyn ensimmäisen aikakauden, joka tunnetaan myös nimityksellä ’tekoälyn kevät’ (AI Spring).

Yhtenä ensimmäisistä nimenomaan ihmisten välisen keskustelun simuloimiseen tarkoitetuista ohjelmista, jota nykypäivän chatbotit sekä muut virtuaaliset assistentit voivat pitää varhaisena edeltäjänään, pidetään ELIZAA. ELIZA on Joseph Weizenbaumin vuosien 1964—1967 aikana MIT:ssä (Massachusetts Institute of

Technology) kehittämä, luonnollisen kielen prosessointiin pohjautuva, ohjelmisto, joka oli myös yksi ensimmäisistä Turingin testin yrittämiseen pystyvistä (Haenlein & Kaplan 2019). ELIZAn alkuperäisenä käyttöliittymänä toimi elektroninen kirjoituskone, josta ohjelma pystyi lukemaan käyttäjän syötteen sekä kirjoittamaan käyttäjälle vastauksen (Weizenbaum 1966). Ohjelmasta on myös jälkikäteen tuotettuja implementaatioita, joilla ELIZAA pystytään käyttämään koneen monitorin välityksellä (kuva 1).

```

Welcome to
          EEEEE LL      IIII ZZZZZZ  AAAAA
          EE      LL      II      ZZ      AA  AA
          EEEEE LL      II      ZZZ      AAAAAA
          EE      LL      II      ZZ      AA  AA
          EEEEE LLLLLL IIII ZZZZZZ  AA  AA

Eliza is a sock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:

```

KUVA 1. Weizenbaumin (1966) kuvaamaan tyypilliseen keskusteluun pohjautuva esimerkkikeskustelu (Wikipedia n.d.).

Tekoälyn kehityksen osalta hiljaisempien 1970—1990-lukujen aikana koettiin kaksi 'tekoälyn talvea' (AI Winter), jotka vaikuttivat merkittävästi tekoälykehityksen edistymiseen. Kyseisen ajanjakson aikana toteutetuissa tekoälysovelluksissa siirryttiin enemmän kehittämään johonkin haluttuun aiheeseen erikoistuvia sovelluksia yleisälykkyyden sijasta. Tätä ilmiötä edesauttoi asiantuntijajärjestelmien (expert systems) käytön yleistyminen 1980-luvulla, joka johtikin hetkelliseen nousuun kiinnostuksessa tekoälykehitykseen. Asiantuntijajärjestelmien tekoäly perustui "jos-niin"-päätelymallin mukaisiin käskyihin, joihin ihmisälykkyyden oletettiin perustuvat (Haenlein & Kaplan 2019). Tämä päätelymalli ei kuitenkaan ollut kovin pätevä sovelluksissa, jotka eivät seuranneet kyseistä rakennetta, joka lopulta asiantuntijajärjestelmien suorituskyvyllisten puutteiden lisäksi lopulta laski niiden suosiota (Lutkevich 2022).

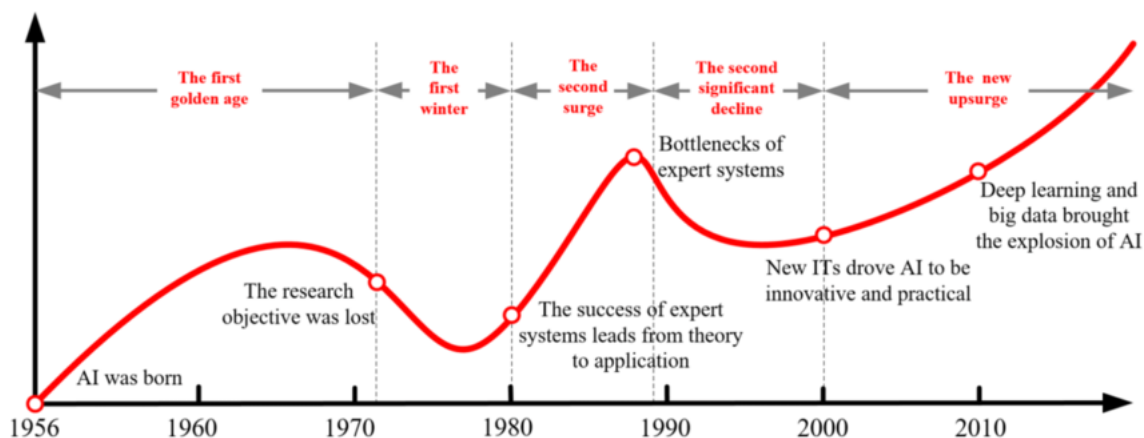


Yhtenä asiantuntijajärjestelmien päättelytavalle sopivana sovelluksena pidettiin lautapelejä. Tällä saralla kolme tunnetuinta tapausta ovat backgammon, shakki sekä Go. BKG 9.8 -nimeä kantava Hans Berlinerin kehittämä ohjelma oli ensimmäinen tekoäly, joka pystyi päihittämään minkään kortti- tai lautapelin senhetkisen maailmanmestarin (Berliner 1980). Kyseinen tekoäly voitti silloisen backgammonin maailmanmestarin Luigi Villan vuonna 1979. Shakin osalta kyseinen tavoite saavutettiin vuonna 1997, kun IBM:n kehittämä ohjelma Deep Blue päihitti shakin maailmanmestarin Garry Kasparovin (Haenlein & Kaplan 2019). Vielä edeltävänä vuonna 1996 Kasparov oli voittanut Deep Bluen.

Ammattilaispelaajan päihittämistä Go-lautapelissä saatiin odottaa vuoteen 2015. Tällöin Google DeepMindin kehittämä AlphaGo päihitti ammattilaispelaajan Fan Huiin ilman tasoituksia. Seuraavana vuonna 2016 AlphaGo voitti silloisena maailmanmestarina pidetyn Lee Sedolin. (Google Deepmind n.d.) AlphaGo ei kuitenkaan enää pohjautunut asiantuntijajärjestelmiin, vaan siinä hyödynnettiin neuroverkkoja, tarkemmin ottaen syväoppimista (Haenlein & Kaplan 2019). Näihin teknologioihin perustuvat myös lähes kaikki nykypäiväiset tekoälyn ja koneoppimisen sovellukset, kuten datankäsittely, hahmontunnistus (Pattern Recognition) sekä generatiiviset tekoälymallit (Generative AI).

### **2.1.1 AI Winter**

Tekoälyn kehityskaari ei kuitenkaan aina ole ollut pelkkää ylämäkeä, vaan matkalla on esiintynyt myös kehityksen osalta hiljaisempia aikakausia (kuva 2). Näitä kyseisiä ajanjaksoja kutsutaan nimityksellä AI Winter, 'tekoälyn talvi'. Termin alkuperä juurtaa vuonna 1984 pidetystä AAAI:n kokouksesta, jossa Marvin Minsky sekä Roger Schank kuvailivat termillä silloista mahdollista edessä olevaa kehityksen kuivaa kautta (Umbrello 2021). Tähän päivään mennessä katsotaan tapahtuneen kaksi tekoälyn talvea. Vaikkakin näiden ajanjaksojen rajat ovat häilyviä sekä lähdekohtaisia, katsotaan näiden talvien sijoittuvan ajanjaksoille 1974—1980 sekä 1987—1993 (Umbrello 2021).



KUVA 2. Tekoälyn kehityksen aikakaudet suuntaa antavalla kuvaajalla esitettyinä (Wang, Liu, Liu & Tao 2021).

Näihin talviin johtaneita syitä on monia, joita ovat muun muassa

- rahoitusten vähentäminen tai lakkauttaminen, kun odotetulle kehitykselle ei ole riittävää näyttöä
- kehityksessä sekä toteutuksessa käytettävien laitteiden sekä asiantuntija-järjestelmien suorituskyvylliset puutteet
- tekoälyn kehittäjien antamien ylilupauksien sekä käyttäjien liian korkeiden odotusten toteutumattomuus
- median paisuneet mainos- ja kritiikkipuheet
- motivaation ja uskon hiipuminen tekoälyn kehitykseen. (Lu 2022; Umbrello 2021.)

## 2.2 Tekoälyn määritelmä

Terminä tekoäly on todella laaja käsite, jonka alle mahtuu useita erilaisia osalueita nykypäiväisten teknologia-, tiede- ja tietotekniikka-alojen saralla. Tämän takia myös tekoälyn tarkka määrittely ei ole niin yksiselitteistä. Tekoälyn ollessa sen verran laaja käsitteenä, on sille myös olemassa useita eri määrittelyjä erilaisine kriteereineen. Useat eri lähteet ja tahot määrittelevät sen hiukan eri lailla toisiinsa verrattuna.

Yksinkertaisuudessaan sekä jonkin verran yleistettynä tekoälyllä tarkoitetaan koneiden kykyä suoriutua sille annettavista tehtävistä, joiden suorittaminen vaatii

ihmisten omaavaa älykkyyttä vastaavien taitojen, kuten päättelykyvyn, pulmanratkonnan tai valinnanteon, simulointia ja hyväksikäyttöä. Määritelmään sisältyy myös koneen kykenevyys oppia, sopeutua sekä kehittyä sille annettuun tietoon sekä edeltäviin toimintoihin pohjautuen. Kyseistä koneiden älykkyyttä voidaan mitata hyödyntämällä erilaisia menettelyjä tai testejä, kuten Turingin testiä.

DSRPAI:n, josta myös tekoäly terminä on peräisin, ehdotuksessaan McCarthy, Minsky, Rochester ja Shannon (1955) määrittelevät tekoälyn tutkimuksen ensimmäistä kertaa seuraavasti: "The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." (McCarthy, Minsky, Rochester & Shannon 1955.) McCarthyn (2007) uudemman määritelmän mukaan tekoäly on tiedettä ja tekniikkaa älykkäiden koneiden, erityisesti älykkäiden tietokoneohjelmien, valmistamisesta.

Taloudellisen yhteistyön ja kehityksen järjestö OECD taas määrittelee AI-järjestelmän paljon lakipohjaisemmin. Grobelnik, Perset ja Russell (2024) kertovat OECD:n määritelmän olevan seuraava:

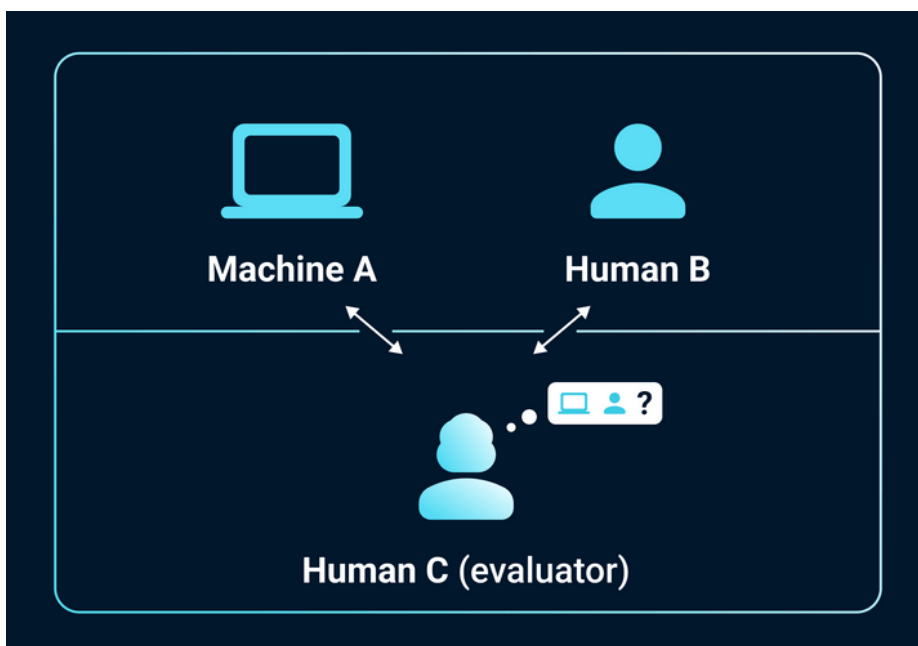
An AI system is a machine-based system that, for explicit or implicit objectives, infers, from the input it receives, how to generate outputs such as predictions, content, recommendations, or decisions that can influence physical or virtual environments. Different AI systems vary in their levels of autonomy and adaptiveness after deployment. (Grobelnik, Perset & Russell 2024.)

Tämän kaltainen laillisempi määritelmä pohjautuu siihen, että kyseistä määritelmää on mahdollista käyttää lakiasetusten ohjeistuksena. Yhtenä esimerkkinä Euroopan Parlamentin asettama tekoäylaki (Artificial Intelligence Act) pohjautuu kyseiseen OECD:n tarjoamaan määritelmään (Coraggio 2023).

### 2.2.1 Turingin testi

Alan Turingin vuonna 1950 julkaistusta artikkelista "Computing Machinery and Intelligence" peräisin oleva Turingin testi on koneiden älykkyyden testaamiseen

kehitetty menettelytapa. Testissä toteutetaan asettelu (kuva 3), jossa testin arvioija (kuvassa Human C) on eristetty testattavista osapuolista (kuvassa Machine A sekä Human B). Testissä arvioijalle ei kerrota, kumman testattavan osapuolen kanssa hän keskustelelee. Tekstipohjaisten kysymysten ja vastausten pohjalta testin arvioijan tehtävänä on tunnistaa, keskusteleeko hän testattavan koneen vai ihmisen kanssa. Jos arvioija ei pysty onnistuneesti erottamaan, kumman keskustelun hän kävi testattavan koneen kanssa, voidaan kyseinen kone luokitella Turingin testin osalta älykkääksi.



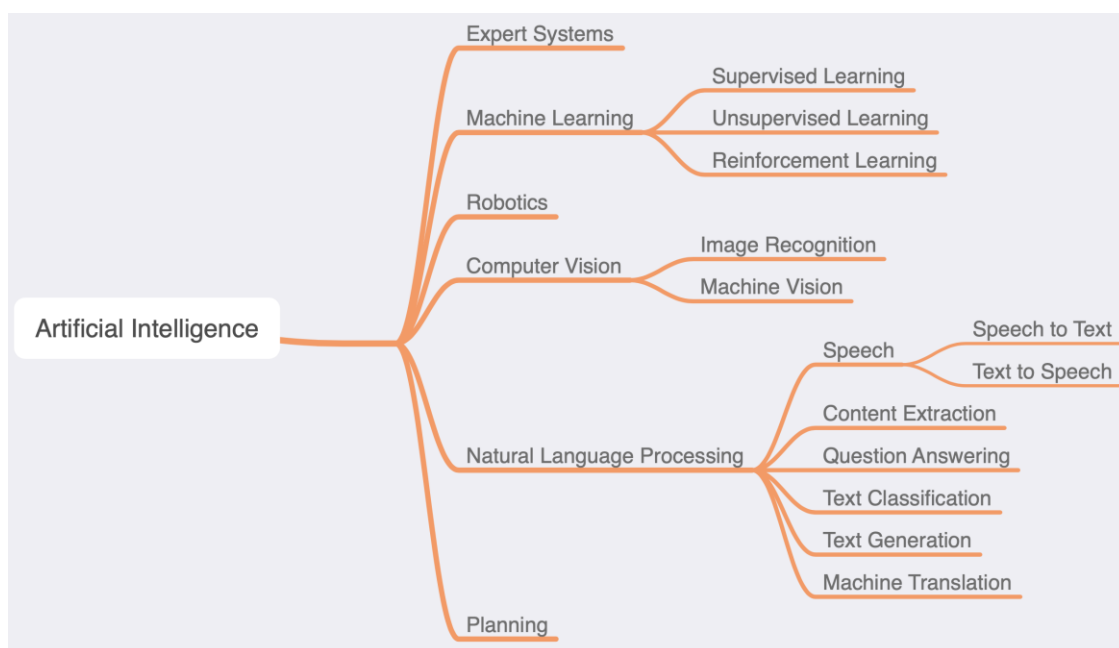
KUVA 3. Turingin testin asettelu (Altmann 2023).

### 3 TEKOÄLYN TEKNOLOGIA

#### 3.1 Tekoälyn osa-alueet

Tekoäly voidaan karkeasti jakaa kuuteen eri osa-alueeseen, joiden alle loputkin tekoälyn teknologioista saadaan ryhmiteltyä (kuva 4). Näitä osa-alueita ovat

- luonnollisen kielen käsittely (NLP)
- koneoppiminen (ML)
- konenäkö (CV)
- asiantuntijajärjestelmät
- robotiikka
- suunnittelu. (Rathod 2020.)



KUVA 4. Tekoälyn osa-alueita (Rathod 2020).

Robotiikassa tekoäly tulee kuvioihin siinä vaiheessa, kun kyseiselle robotille annetaan mahdollisuus ajatella sekä suorittaa tehtäviä itsenäisesti. Suoritettavien tehtävien haasteellisuuden kasvaessa myös tekoälyn sekä koneoppimisen rooli kasvavat. Hyvänä esimerkkinä näiden osa-alueiden hyödyntämisestä on tunnettu Sophia-robotti, jota kykenee sosiaaliseen kanssakäymiseen ihmisten kanssa. (Rathod 2020.)

Automatisoidun suunnittelun päämääränä on nimensä mukaisesti suunnitella sekä optimoida tapahtumaketjuja ja niiden kulkuun johtavia askelia tiettyjen annettujen kriteerien tai parametrien pohjalta. Suunnittelun osa-alueita voidaan hyödyntää esimerkiksi miehittämättömien kulkuneuvojen reitinvalintajärjestelmissä.

Konenäön avulla laite voi tulkita haluttua informaatiota sille syötettävistä kuvista tai videoista. Konenäöllä voidaan tunnistaa, luokitella tai seurata tietyt kriteerit tai kategoriat täyttäviä hahmoja tai objekteja (kone osaa erottaa ihmiset muusta kuvasyötteen sisällöstä). Konenäköä käytetään muun muassa ajoneuvojen reaaliaikaisessa ympäristön seurannassa, sisällön kategoriointiin sekä järjestämiseen liittyvissä applikaatioissa, laitteiden kasvotunnistuskäytöksissä, lisätyn todellisuuden (Augmented Reality) käyttötarkoituksissa ja tuotannon laadunvalvonnassa. Syväoppimista hyödyntämällä konenäöllä pystytään saavuttamaan parempia ja tarkempia lopputuloksia. (Azure n.d.)

### **3.1.1 Koneoppiminen**

Koneoppiminen on yksi suurimmista tekoälyn osa-alueista, jonka teknologioita käytetään myös lähes kaikilla muilla osa-alueilla. Koneoppimisella, kuten nimestäkin voidaan päätellä, tarkoitetaan tekoälyn haaraa, jonka tarkoituksena on annettua dataa sekä tiettyjä opetusmenettelyjä ja algoritmeja hyödyntäen mahdollistaa koneelle oppimismalleja, jotka imitoivat ihmisten tapaa oppia ja kehittyä (IBM n.d.a).

Koneoppiminen voidaan jakaa neljään eri pääoppimismenettelyyn: valvottuun (supervised), valvomattomaan (unsupervised), puoliohjattuun (semi-supervised) sekä vahvistavaan (reinforcement) oppimiseen. Tietyn oppimistavan valitseminen määritellään yleensä käyttökohteen tarpeiden mukaisesti. Valvotussa oppimisessa koneoppimismalli opetetaan valmiiksi merkityllä ("sisältää halutun vastauksen", kuten kasvilajikkeen kuvassa kasvusta) datalla. Tällaisesta datasta oppiessaan malli kykenee ennustamaan lopputuloksia ja luokittelemaan halutun tiedon sille syötetystä uudesta merkitsemättömästä datasta, mikä onkin kyseisen oppimistavan vahvuus. (Coursera 2024.)

Valvomattomassa oppimisessä mallille syötetään merkitsemätöntä dataa, josta mallin algoritmien tehtävänä on löytää datapisteitä yhdistäviä piirteitä. Tästä syystä valvoton oppiminen soveltuukin suurille määrille ”raakaa”, merkitsemätöntä dataa, josta halutaan tehokkaasti löytää yhtäläisyyksiä (Coursera 2024). Valvomatonta oppimista voidaan myös jalostaa pienellä määrällä merkittävää dataa, milloin se saadaan soveltumaan enemmän valvotun oppimisen kaltaisiin sovelluksiin. Tällöin kyseessä on puoliohjattu oppiminen. Tätä oppimistapaa käytetään yleensä tilanteissa, joissa suuren merkityn datamäärän tuottaminen tai hankkiminen ei ole mahdollista tai kannattavaa (IBM n.d.a).

Vahvistava oppiminen poikkeaa muista oppimistavoista siinä, että mallia ei opeteta käyttämällä testidataa. Sen sijaan vahvistava oppiminen perustuu yrityserhdys-oppimistyyliin, jossa mallia palkitaan sen onnistumisen perusteella ja näin se kehittyy myös ajan myötä tekemään tehokkaampia päätöksiä. Vahvistavaa oppimista käytetäänkin yleensä tilanteissa, joissa algoritmin vaiheiden tehokkuus on tärkeintä. (Coursera 2024.)

Koneiden oppiminen perustuu erilaisten algoritmien käyttöön. Nämä algoritmit sisältävät koneiden tarvitsemat käskyt haluttujen tehtävien suorittamiseksi. Algoritmit ovatkin juuri se, mitä edellä mainittujen oppimismenetelmien sekä testidatan avulla parannellaan, eli koneoppimismallit ovat koulutettuja algoritmeja (Coursera 2024). Näitä algoritmeja on myös mahdollista ketjuttaa yhteen muodostamaan verkostoja, joista käytetään nimitystä neuroverkko. Neuroverkoissa algoritmit liittyvät toisiinsa painotettujen yhteyksien avulla, joita säädetään neuroverkon oppimisvaiheessa.

Kun neuroverkon kerrosten syvyys kasvaa yli perinteisen kolmen kerroksen (1 syötekerros, 1 piilokerros ja 1 ulostulokerros), aletaan käyttää termejä syväoppiminen ja syvät neuroverkot. Mitä useampia kerroksia neuroverkkoon liitetään, sitä monimutkaisempia tehtäviä sen avulla pystytään suorittamaan, ja myös ihmisen tarve puuttua datan käsittelyyn vähenee. Syvät neuroverkot soveltuvat esimerkiksi konenäköön, luonnollisen kielen käsittelyyn sekä varsinkin generatiivisen tekoälyyn liittyviin tehtäviin.

### 3.1.2 Luonnollisen kielen käsittely

Luonnollisen kielen käsittely on tekoälyn osa-alue, jossa käsitellään koneiden kykyä ymmärtää ja luoda ihmisten käyttämää, suullista ja kirjallista, eli luonnollista kieltä. Luonnollinen kieli sisältää paljon erilaisia muuttujia, jotka pitää huomioida sitä käsitellessä. Tällaisia huomioon otettavia seikkoja ovat muun muassa eri kielet, slangit ja aksentit, sanojen poikkeava ääntäminen, referenssit ja kontekstit, kiertoilmaukset ja metaforat, sarkasmi, sanojen taivutukset sekä erisnimiluokittelu (IBM n.d.b). Luonnollisen kielen tekoälymallien tulee sisäistää kaikki vastaavat muuttujat luonnollisessa kielessä, jotta koneen ymmärtäminen sekä varsinkin uuden sisällön generoiminen olisi uskottavalla ja hyödyllisellä tasolla.

Luonnollisen kielen käsittely tarvitaan yhä kasvavassa määrässä käyttökohteita sitä mukaan, kun tekoälyn käyttö ja ihmissyötteen käsittely yleistyvät. Luonnollisen kielen käsittelyä hyödynnetään käyttökohteissa kuten puheen tunnistamisessa ja luomisessa, kielen kääntämissovelluksissa, virtuaaliassisteissa sekä tekstinkäsittelysovelluksissa kuten tekstin tiivistäjissä tai luokittelijoissa (IBM n.d.b). Myös nykyaikaiset generatiivista tekoälyä hyödyntävät chatbotit ja kehitystyökalut ovat vahvasti tekemisissä luonnollisen kielen käsittelyn kanssa.

### 3.2 Suuret kielimallit

Suurilla kielimalleilla tarkoitetaan tekoälymalleja, joiden tarkoituksena on ymmärtää, käsitellä sekä luoda ihmiskieltä. Ne kuuluvatkin tämän myötä NLP-osa-alueeseen. Suurten kielimallien avulla pyritään parantamaan tekoälyn kykyä suoriutua kielen mallinnukseen liittyvissä tehtävissä. Kielen mallinnus on yksi NLP:n osa-alueista, jossa luotavien mallien tavoitteena on tiettyjen token-ketjujen toteutumisen ennustaminen käsiteltävästä sanastosta tai kielestä riippuen. Termillä token viitataan tekstin pienimpään semanttiseen osaan, joka voi muodostua sanasta tai vain sanan osasta. (Ozdemir 2024, luku 1.1.) Tokeneita käytetäänkin perusyksikkönä suurten kielimallien sekä generatiivisen tekoälyn sovelluksissa.

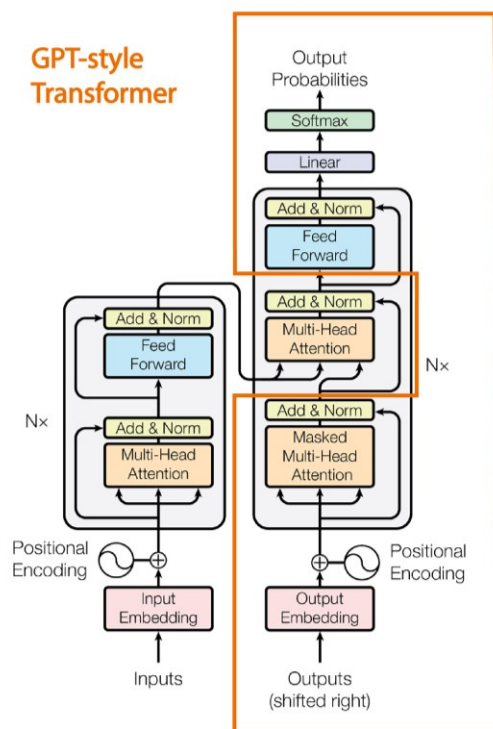
Suurten kielimallien kokoa voidaan vertailla muun muassa niiden sisältämien parametrien tai mallien kouluttamiseen käytetyn koulutusdatan (mitattu tokeneissa)



määrällä. Nykyaikaisten suurimpien kielimallien parametrit sekä koulutusdatan tokenit ovat satojen miljardien tai biljoonien luokkaa. Esimerkiksi Metaan Llama 3 -kielimalli on opetettu yli 15 biljoonalla tokenilla ja kyseinen malli sisältää tällä hetkellä päälle 400 miljardia parametria (Meta 2024).

Suurin osa suurista kielimalleista pohjautuvat syväoppimismalliin, josta käytetään nimitystä transformer-arkkitehtuuri. Alkuperäinen transformer-arkkitehtuuri kuvattiin Googlen työryhmän toimesta vuonna 2017 julkaistussa artikkelissa ”Attention Is All You Need”, joka koostuu kahdesta pääkomponentista: dekooderista, jonka vahvuus on tekstin generoiminen, sekä enkooderista, joka puolestaan suoriutuu paremmin tekstin ymmärtämisessä (Vaswani ym. 2017).

Jotkin kielimallit kuitenkin hyödyntävät vain jompaakumpaa alkuperäisen transformer-arkkitehtuurin komponenteista. Esimerkiksi OpenAI:n kehittämissä GPT-kielimalleissa käytetään vain transformerin dekooderiosaa. (Stollnitz 2023.) Kuvassa 5 on nähtävissä koko alkuperäinen transformer-arkkitehtuuri sekä GPT-mallien hyödyntämä osio.



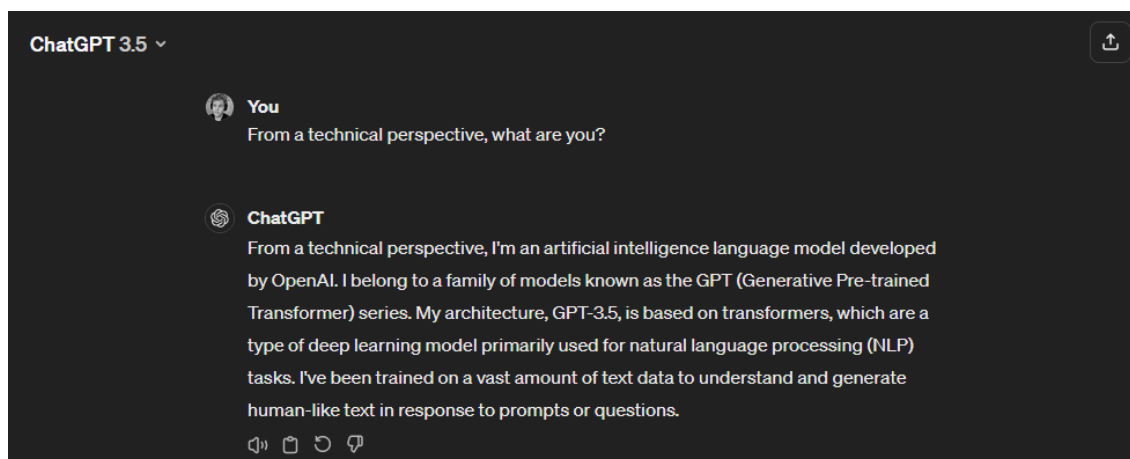
KUVA 5. Alkuperäinen transformer-arkkitehtuuri (Stollnitz 2023).

### 3.3 Tekoälyn sovellukset

#### 3.3.1 ChatGPT

ChatGPT on OpenAI:n kehittämä tekoälypohjainen chatbot-alusta, jonka avulla voidaan käyttää OpenAI:n tarjoamia GPT-tekoälymalleja. Se julkaistiin käyttäjien saataville vuoden 2022 marraskuussa, josta lähtien se on kasvattanut suosiotaan sekä vastaavanlaisten tekoälysovellusten tuntemusta jatkuvasti laajenevan käyttäjäkunnan piireissä.

ChatGPT-chatbotille voidaan syötteenä esittää erilaisia pyyntöjä tai kysymyksiä, joihin tekoälyn avulla generoidaan vastaukset. Kuvassa generoidussa syötteessä ChatGPT kertoo perustuvansa GPT-transformer-malleihin, jotka ovat luonnollisen kielen käsittelyssä käytettyjä syväoppimisen malleja. ChatGPT:n tavoitteena on ymmärtää ja generoida ihmistä muistuttavaa tekstiä vastauksena syötteisiin tai kysymyksiin. (Kuva 6.)



KUVA 6. ChatGPT:n vastaus kysymykseen ”From a technical perspective, what are you?”

ChatGPT:stä on tarjolla useampi eri versio, jotka poikkeavat toisistaan hinnoittelussa sekä saatavilla olevissa ominaisuuksissa. ChatGPT:n ilmaisversio perustuu tällä hetkellä pääosin GPT-3.5-kielimalliin, mutta OpenAI on juuri alkanut tarjoamaan uutta GPT-4o-kielimallia rajoitetuin ominaisuuksin ilmaiskäyttäjille (OpenAI

2024). Tällaisia ominaisuuksia edeltävään GPT-3.5-malliin lisäten ovat muun muassa netin selaus, data- ja tiedostanalyysi sekä konenäkö (Pricing n.d.).

ChatGPT:n maksulliset versiot perustuvat neljännen sarjan GPT-malleihin GPT-4 ja GPT-4o. Näitä malleja käytettäessä käyttäjän syöte voi olla multimodaalista, eli tekoälymallille pystytään syöttämään useita syötemuotoja kerrallaan, kuten ääntä, videokuvaa sekä tekstiä. Malli pystyy myös vastaavasti vastaamaan syötteeseen usealla eri syötemuodolla. Kuvien generointi, jossa hyödynnetään OpenAI:n DALL·E 3 -mallia, kuuluu maksullisten ChatGPT-versioiden ominaisuuksiin. Tämän lisäksi maksullisten versioiden ominaisuuksiin kuuluvat esimerkiksi suuremmat viestirajat, personoidumpien ja kohdennetumpien GPT-mallien luonti ja käyttö, malleja ja ominaisuuksia koskevien rajoitusten vähentäminen sekä yksityisyys-, turvallisuus-, hallinto- ja asiakaspalveluetuja (Pricing n.d.).

OpenAI tarjoaa myös saatavilla oleville GPT-malleille API-rajapintoja, joiden avulla esimerkiksi sovelluskehittäjät voivat integroida näitä malleja käytettäväksi sovelluksissaan. Nämä kyseiset rajapinnat ovat maksullisia. (OpenAI n.d.)

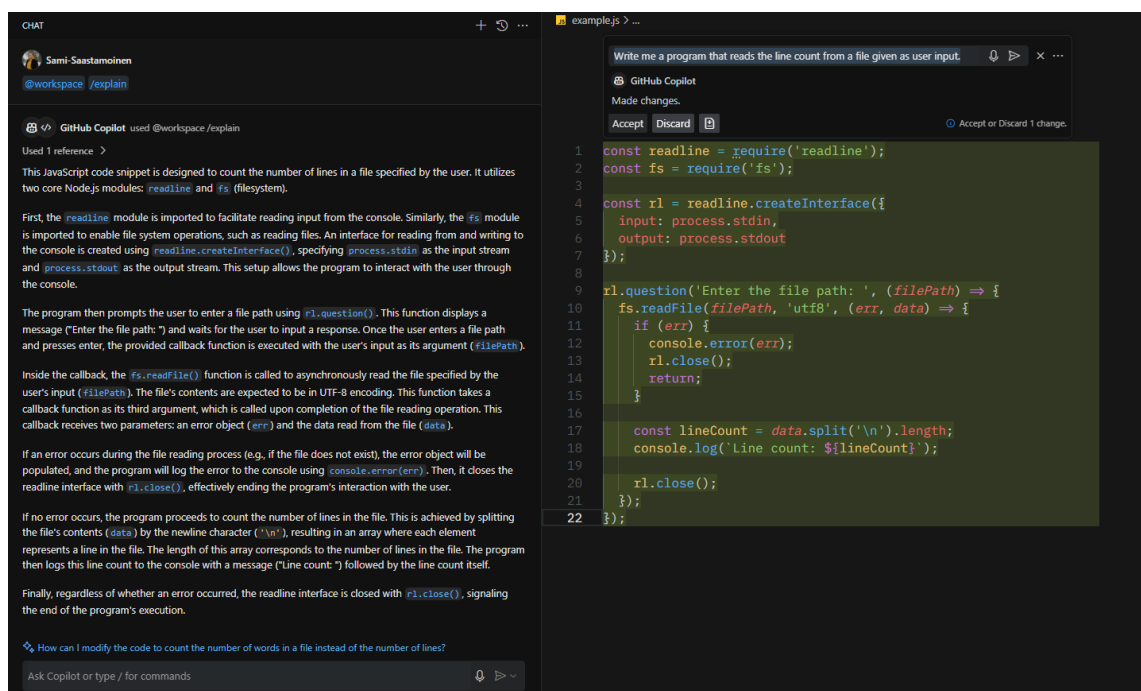
### 3.3.2 GitHub Copilot

GitHub Copilot on GitHubin sekä OpenAI:n yhteistyönä toteutettu tekoälypohjainen, kehitysavustusta tarjoava työkalu, joka julkaistiin vuonna 2021. Copilotin tarkoituksena on kasvattaa kehittäjien tuottavuutta sekä auttaa nopeuttamaan sovelluskehitystä ja koodausta ylipäättään (Copilot n.d.). Tähän tavoitteeseen Copilot pyrkii tarjoamallaan ominaisuuksilla, kuten reaaliaikaisesti generoiduilla kontekstipohjaisilla koodin täydennysehdotuksilla, laajalla ohjelmointikielituella, integraatioilla suoraan käytettävään koodieditoriin ja terminaaliin sekä avustavalla chatbotilla.

Copilotin tarjoamat koodiehdotukset perustuvat OpenAI:n GPT-3-pohjaiseen Codex-kielimalliin, joka on koulutettu erityisesti ymmärtämään sekä generoimaan koodisisältöä. Codex-mallin koulutusdatassa on hyödynnetty julkisesti saatavilla olevia lähdekoodeja useilla eri ohjelmointikielillä, kuten GitHubista löytyviä julkisia projekteja. Koodiehdotuksia voidaan generoida kooditiedoston aiemman sisällön

perusteella, tai Copilotille voidaan kommenttina antaa halutun generoitavan koodin kuvaus ja parametrit.

Copilot Chat -chatbot taas perustuu GPT-4-kielimalliin ja on rajattu vastaamaan ohjelmointiin liittyviin kysymyksiin. Chat kykenee esimerkiksi selittämään koodia tai terminaalin syötettä, korjaamaan koodista löytyviä virheitä, luomaan testejä tai vastaamaan projektiin liittyviin kysymyksiin. Kontekstina Chatille voidaan valita kaikkea valikoidun koodiosion ja koko projektin väliltä. Kuvassa 7 kuvattuna Copilotin generoima koodi annetun syötteen perusteella sekä Chatin antama selitys kyseiselle koodille.



KUVA 7. GitHub Copilotin koodigenerointi (oik.) sekä Copilot Chatin selitys.

Copilotista on saatavilla kolme eri tasoista versiota: Individual, Business ja Enterprise. Yksilöille tarkoitettuun Individual-tasoon nähden Business-tason versio sisältää organisaatioille hyödyllisiä ominaisuuksia, kuten käyttäjien hallinnan, SSO-todennuksen, immateriaalioikeuksia koskevat vakuudet sekä data käyttämättömyyden koulutusdatana oletusarvoisesti. Yrityksille tarkoitettua Enterprise-versiolla saa näiden lisäksi pääsyn yritykselle mukautettuihin keskusteluihin ja vastauksiin, verkkohakuihin, kattavampiin versionhallinnan integraatioihin sekä hienosäädettyihin kielimalleihin. Tasojen kuukausihinnat alkavat 10 dollarista käyt-

täjää kohden, mutta todennetut opiskelijat, opettajat ja suosittujen avoimen lähdekoodin projektien ylläpitäjät voivat käyttää Copilot Individual -tasoa ilmaiseksi. (Copilot n.d.)

## 4 PROJEKTIEEN RAJAUS JA TEKNOLOGIAT

Tässä työssä toteutettavat ohjelmistoprojektit rajattiin muistuttamaan toisiaan sekä aiheen että käytettävien teknologioiden osalta. Ohjelmistoprojektien aiheeksi valikoitiin web-pohjainen datan visualisointityökalu nimeltä Disc Tracker, jonka avulla voidaan käsitellä sekä näyttää sille syötettävää frisbeegolfin liittyvää tilastodataa. Työkalu koostui kolmesta sivusta:

- Työkalun kotisivulla käyttäjä voi syöttää visualisoinnissa käytettävän JSON-tiedoston (ECMA-404 2017). Kyseisen tiedoston sisällön rakenne myös validoidaan kyseisellä sivulla.
- Tilastosivulla syötettyä dataa visualisoidaan erilaisten diagrammien, kuten pylväs-, viiva- ja ympyrädiagrammien, avulla.
- Formaattisivulla käyttäjälle esitetään JSON-tiedoston tarvittava rakenne, jota hyödynnetään myös tiedostojen validoinnissa.

Molemmat projektit luotiin sekä alustettiin samalla tavalla, jolloin projektien aloitustämäärittelyt olivat samat. Työn pohjana käytettiin Next.js-sovelluskehystä. Next.js-kehityksen valinta perustui siihen, että se oli ennestään tuntematon muuttaja, jolloin projekteihin saatiin myös näkökulma uusien teknologioiden käytöstä. Työkalun tyylittelyssä hyödynnettiin Tailwind CSS -sovelluskehystä (Tailwind n.d.). Muut projekteissa tarvittavat kirjastot valikoitiin projektikohtaisesti joko oman tiedonhaun tai tekoälyn päätöksen perusteella.

Työssä käsiteltävistä ohjelmistoprojekteista ensimmäisenä toteutettiin perinteiseksi luokiteltu ohjelmistoprojektin toteutustapa, jonka jälkeen vasta toteutettiin tekoälyavusteinen ohjelmistoprojekti. Työhön valittiin kyseinen toteutusjärjestys, jotta tekoälyavusteisen projektin toteutukset tai sisältö eivät olisi vaikuttaneet perinteiseksi luokitellussa projektissa käytettyihin toteutustapoihin tai siitä saatuihin lopputuloksiin. Tällä toteutusjärjestyksellä pystyttiin myös tehokkaammin ja kriittisemmin seuraamaan tekoälytyökalujen toteuttamaa sisältöä, huomioimaan mahdollisia puutteita tai virheitä kyseisessä tuotetussa sisällössä sekä tarvittaessa ohjastamaan käytettäviä tekoälytyökaluja oikeaan suuntaan, jotta kyseisellä projektin toteutustavalla päästäisiin mahdollisimman lähelle samaa lopputulosta

kuin perinteisessäkin ohjelmistoprojektissa saatujen tulosten vertailukelpoisuuden parantamiseksi.

#### **4.1 Perinteinen ohjelmistoprojekti**

Perinteisessä projektin toteutustavassa ei tultu käyttämään, ainakaan tietoisesti tai omatoimisesti, tarjolla olevia tekoälytyökaluja tai muutakaan tekoälyn tuottamaa sisältöä. Sen sijaan perinteisen toteutustavan projektin kehittämisen tukena käytettiin verkossa julkisesti saatavilla olevia tietolähteitä ja materiaaleja, kuten teknologioiden ylläpitäjien tai muiden tahojen tarjoamia dokumentaatioita ja opetusmateriaaleja käsittelyssä olevista aiheista, sekä foorumeja ja keskustelupalveluita, kuten Stack Overflow tai muut aihekohtaisemmat keskustelupalstat. Verkosta saatavien tietojen tarkoituksena on tukea henkilökohtaista, jo ennestään hankittua, tuntemusta käytettävistä teknologioista sekä ohjelmistoprojektien toteutuskäytänteistä.

#### **4.2 Tekoälyavusteinen ohjelmistoprojekti**

Tekoälyavusteisessa ohjelmistoprojektin toteutustavassa sisällön tuottamiseen tultiin pääasiassa käyttämään julkisesti saatavilla olevia tekoälytyökaluja sekä mahdollisesti myös muita tarjolla olevia tekoäly sivustoja tai chatbotteja, kuten OpenAI:n ChatGPT. Tämän projektin kehitykseen tultiin kuitenkin pääsääntöisesti hyödyntämään GitHub Copilot -tekoälytyökalua, joka tämän työn yhteydessä oli integroituna Visual Studio Code -koodieditoriin ladattavana lisäosana.

Jotta työssä saavutettaisiin tarvittavan selkeät erot toteutettavien projektien toteutustapojen välille, tekoälyavusteisessa projektissa ei tultu käyttämään verkossa saatavilla olevaa informaatiota liittyen käytettäviin teknologioihin tai toteutustapoihin. Tällöin projektissa saatiin selkeämpi kuva siitä, mihin tekoälytyökalut pystyvät toteutettavan sisällön kannalta, jolloin tämän toteutustavan mahdolliset hyödyt sekä haitat korostuivat projektien jälkeen toteutetussa vertailussa perinteiseen toteutustapaan nähden.

## 4.3 React

React on Metan (entinen Facebook) kehittämä ja ylläpitämä avoimen lähdekoodin JavaScript-kirjasto, joka julkaistiin vuonna 2013. React on tarkoitettu frontend-kirjastoksi, jonka avulla voidaan luoda JavaScript-funktioihin perustuvien komponenttien avulla käyttöliittymiä erilaisiin käyttötarkoituksiin sekä web- että natiivipohjaisena (React n.d.).

Web-puolella Reactia käytetään SPA-sovelluksien kehittämiseen. Itsessään Reactilla toteutetaan frontend-komponentteja, joissa JavaScript-funktiot palauttavat JSX-syntaksia vastaavia moduuleita renderöitäväksi sovelluksessa. (Herbert 2023.)

Vaikkakin React on tarkoitettu frontend-kehitykseen, tarjolla olevia kirjastoja sekä sovelluskehysiä, kuten Next.js, hyödyntämällä myös kattavien fullstack-applikaatioiden rakentaminen on mahdollista. (Start a New React Project n.d.)

Natiivissa React-kehityksessä taas hyödynnetään React Native-sovelluskehystä, jossa komponentit renderöivät HTML-pohjaisten web-näkymien sijasta kunkin kehitettävän alustan tarjoamia natiiveja näkymiä sekä komponentteja. Pääasialliset alustat React Native-sovelluksille ovat Android sekä iOS, mutta myös muita alustoja on tuettuna. (React n.d.; React Native n.d.)

### 4.3.1 Next.js

Next.js on Reactille tarkoitettu avoimen lähdekoodin web-applikaatioiden kehittämiseen suunnattu sovelluskehys. Se julkaistiin vuonna 2016 ja on Vercelin kehittämä. Next.js:n tavoitteena on auttaa sovellusten kehittäjiä toteuttamaan tehokkaita ja nopeampia fullstack-applikaatioita (What is Next.js? n.d.).

Next.js:ää käytettäessä sivuston frontend-käyttöliittymät toteutetaan edelleen Reactin komponenteilla, mutta tämän lisäksi Next.js tarjoaa ylimääräisiä opti-



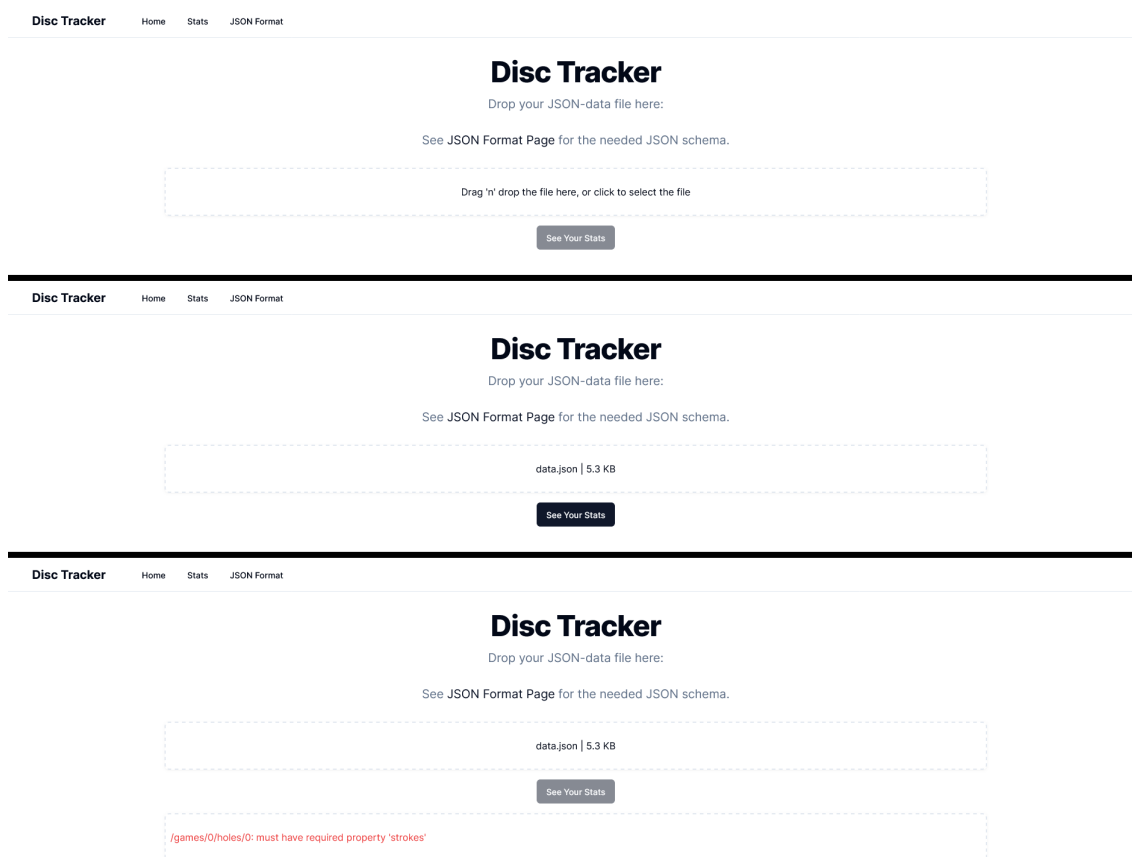
moiteja sekä ominaisuuksia täydentämään kehittämistä, kuten kuvien sekä fonttien automaattisen optimoinnin, tiedostorakennepohjaisen sivujen reitittämisen, asiakas- ja palvelinpuolen komponentit sekä renderöinnin, asynkronisen tietojen haun sekä parannellun tuen TypeScript-ohjelmointikielelle. (Next.js n.d.; Introduction n.d.)

## 5 PROJEKTIEIN TOTEUTUS

### 5.1 Perinteinen ohjelmistoprojekti

Perinteisen projektin työkalussa valmiina komponentteina käytettiin shadcn/ui:n tarjoamia komponentteja. Shadcn/ui ei varsinaisesti ole komponenttikirjasto, vaan kokoelma valmiita komponentteja, joiden koodi on avoimesti käytettävissä projekteissa. Kokoelma sisälsi kaiken tarvittavan sisällön työkalun toteuttamiseen, kuten valmiit komponentit napille, korttielementille, navigointipalkille ja tekstin vieritysalueelle (scroll area) sekä tyylittelyt typografialle. Kyseisten komponenttien valintaan johdatti projekteja edeltävä mielenkiinto käyttöä kohtaan. Shadcn/ui:n komponenttien tyylittely on myös toteutettu Tailwind CSS:n avulla, joka sopii projekteille asetettuun rajaukseen. Tailwindin avulla elementit tyylitellään käyttämällä jo valmiiksi olemassa olevia luokkia sen sijaan, että kaikki käytettävät luokat jouduttaisiin itse toteuttamaan perinteisiin CSS-tiedostoihin (Tailwind n.d.).

Projektissa toteutetun työkalun kotisivulla (kuva 8) tiedostojen vastaanottamiseen käyttäjältä käytettiin react-dropzone-kirjastoa, jonka avulla voidaan luoda projekteihin raahaa ja pudota -syötekenttiä tiedostoja varten. React-dropzonella asetettiin työkalu hyväksymään syötteeksi maksimissaan 1 tiedosto sekä hyväksyttäväksi tiedostomuodoksi '.json'.



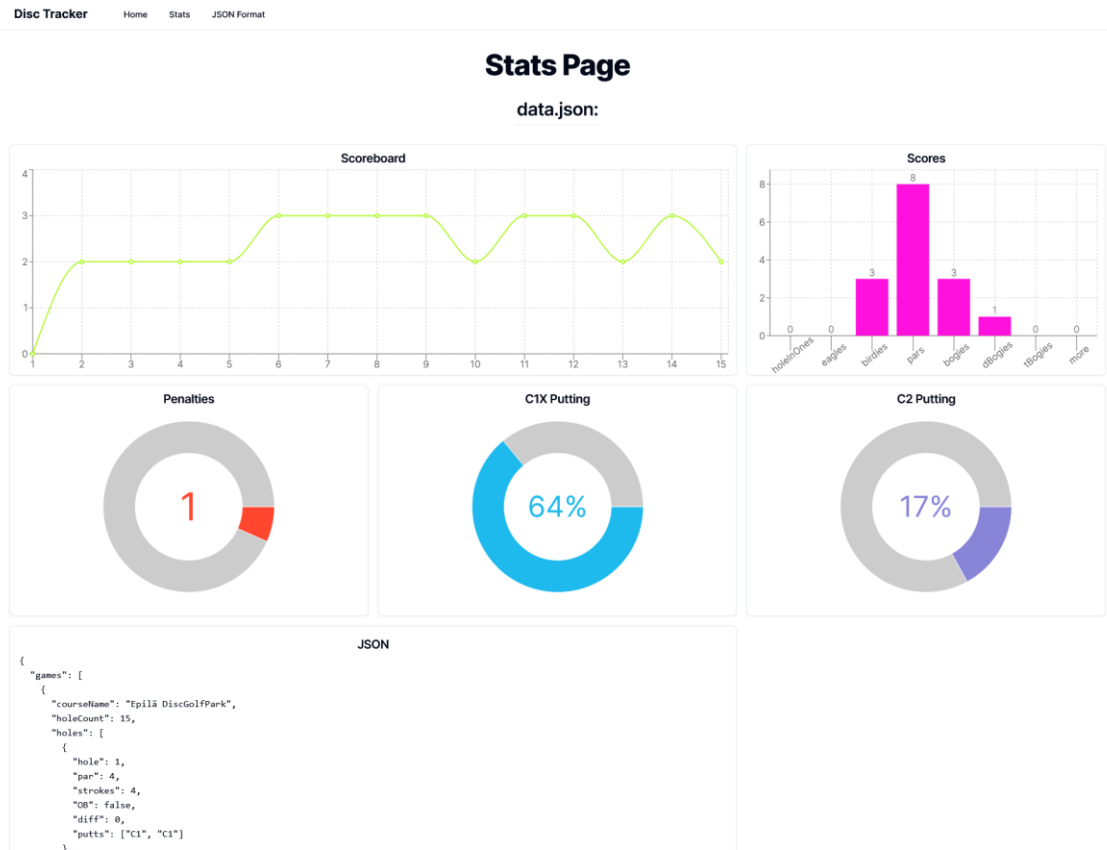
KUVA 8. Perinteisen projektin kotisivu ilman tiedostoa sekä validoinnissa hyväksytyllä ja hylätyllä tiedostolla.

Tiedoston syöttämisen jälkeen sen sisältö ja rakenne validoitiin Ajv-kirjaston (Another JSON Schema Validator) avulla. Ajv:lle tarjotaan haluttu JSON-rakenne kaaviona (schema), johon kirjaston metodit vertaavat syötteenä saadun tiedoston sisältöä. Jos syötettävän tiedoston tiedostotyyppi on oikea sekä sen sisältö vastaa käytettävää kaavaa, antaa työkalu tallentaa tiedoston sisällön selaimen paikallistallenustilaan (local storage) käytettäväksi datan visualisoinnissa. Toteutukseltaan yksinkertaisena lisänä projektiin käytettävä JSON-kaava esitetään käyttäjälle työkalun formaattisivulla (kuva 9).



KUVA 9. Perinteisen projektin formaattisivu.

Perinteisen projektin tilastosivulla (kuva 10) datan visualisoinnissa hyödynnettiin Recharts-diagrammikirjastoa. Recharts osoittautui projektia varten tehdyn tiedonhaun perusteella suosituksi valinnaksi tämän työn rajausta vastaaviin sovelluksiin. Recharts-kirjasto sisälsi työkalun toteuttamiseen tarvittavat diagrammit, joiden käyttö oli suurimmaksi osaksi vaivatonta kirjaston dokumentaation avulla.



KUVA 10. Perinteisen projektin tilastosivu.

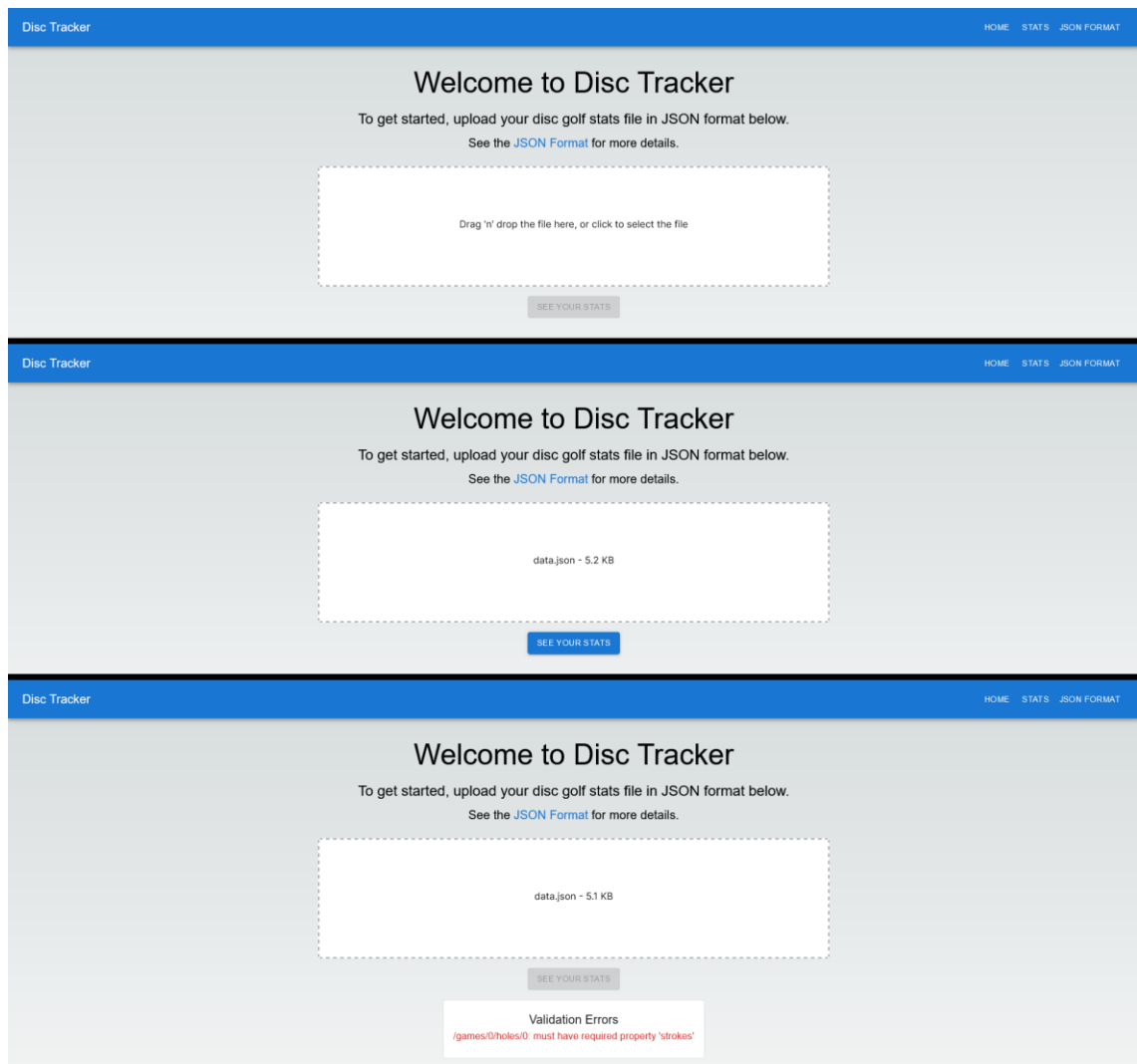
## 5.2 Tekoälyavusteinen ohjelmistoprojekti

Tekoälyavusteinen projekti ohjattiin toteutusvaiheessa muistuttamaan mahdollisimman paljon perinteisen toteutustavan projektia, jättäen kuitenkin tekoälyn päätösvaltaan rajauksen ulkopuolisten tarvittavien kirjastojen valitsemisen. Pääpiirteiltään projektin lopputulos olikin hyvin samankaltainen perinteiseen projektiin verrattuna. Työkalun visuaalisen ulkoasun erot johtuivatkin pääasiassa vain valittujen kirjastojen muuttumisesta.

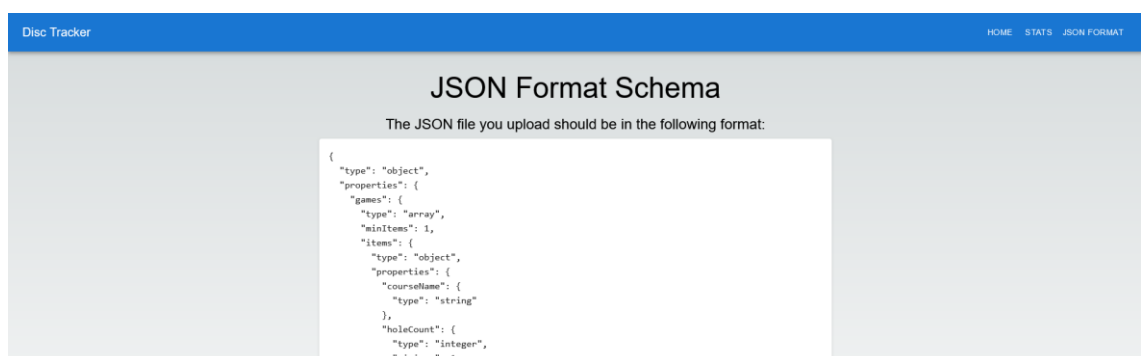
Copilot Chat ehdotti työkalun komponenteissa käytettäväksi Material UI -komponenttikirjastoa, joka mukailee Googlen Material Design -suunnitteluviitteitä. Material UI:n komponentit käyttävät myös tyylittelyssään komponenttien tarjoamia vaihtoehtoja, jolloin myös projektirajauksessa esitetyn Tailwind CSS:n käyttö jäi tekoälyprojektissa vähemmälle.

Päätöstään Copilot Chat perusteli kyseisen kirjaston kattavilla ominaisuuksilla, mukautusvaihtoehdoilla ja dokumentaatiolla sekä laajalla käyttäjäyhteisöllä ja tuella. Copilotille ehdotettiin valintavaiheessa myös erillisesti perinteisessä projektissa käytettyä shadcn/ui:ta, mutta Copilot ei tunnistanut kyseistä komponenttikokoelmaa "tunnetuksi komponenttikirjastoksi JavaScript- tai React-ekosysteemissä". Myöskään sen jälkeen, kun syötteenä tarjottiin linkki kokoelman kotisivuille, Copilot ei osannut antaa kuin sovellettuja, yleispiirteisiä esimerkkejä komponenttikirjastojen käytöstä React-projekteissa.

Käytettyjen komponenttien osalta muutoksia oli vain muutamia. Tekoäly ei hyödyntänyt toteutuksessaan valmista komponenttia tekstin vieritysalueelle, koska sellaista ei ollut sisällytetty Material UI:hin. Lisäyksenä perinteisen projektiin käytössä olivat valmiit komponentit konteille (container ja box), ruudukolle (grid), lausikonille sekä teksteille (typography). Muut kotisivun rakentamiseen tarvittavat kirjastot tekoäly valikoi samoiksi kuin perinteisessäkin projektissa, eli tiedostojen raahaa ja pudota -toiminnallisuus toteutettiin react-dropzone-kirjastolla ja tiedostojen datan validointiin käytettiin Ajv-kirjastoa (kuva 11). Nämä kirjastot Copilot valitsi käyttöönsä ilman tarkempia perusteluja sille syötteenä annettujen toiminnallisuusvaatimusten ja -kuvausten pohjalta. Työkalun formaattisivun (kuva 12) luominen oli tekoälyllekin yksinkertainen tehtävä.



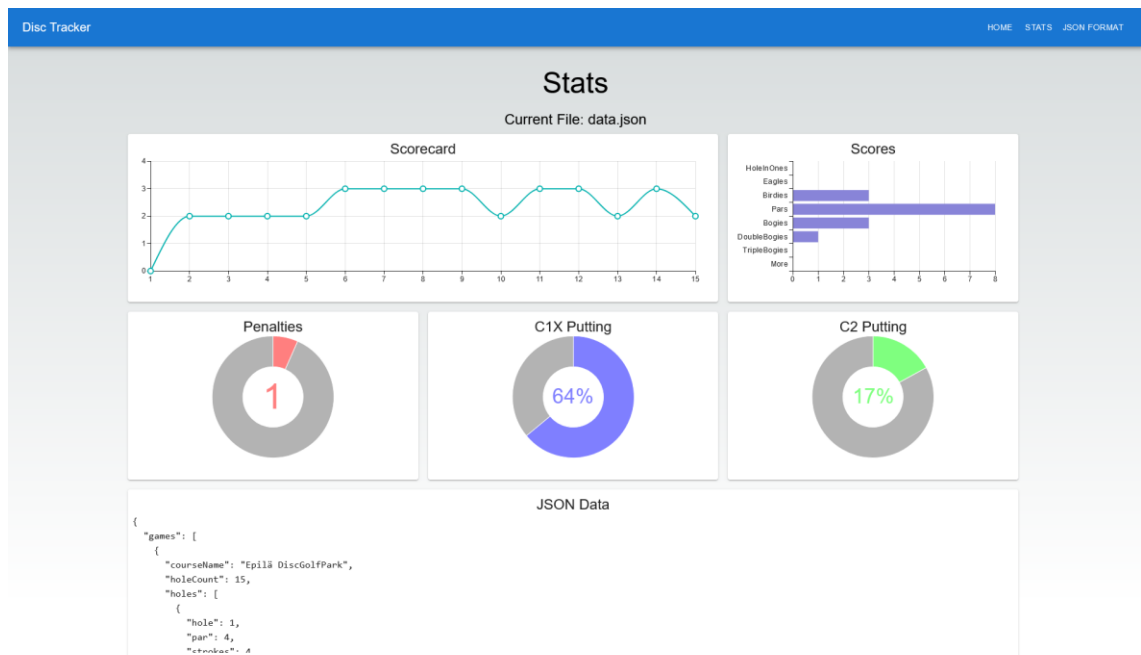
KUVA 11. Tekoälyprojektin kotisivu ilman tiedostoa sekä validoinnissa hyväksytyllä ja hylätyllä tiedostolla.



KUVA 12. Tekoälyprojektin formaattisivu.

Työkalun tilastosivun (kuva 13) visualisoinneissa käytettäväksi Copilot Chat suosittelee MUI (Material UI) X Charts -diagrammikirjastoa. Päätöksen perusteluna oli MUI X Charts -kirjaston diagrammien tyyllittelyn johdonmukaisuus muiden työkalussa käytettävien komponenttien kanssa. Rechartsin tapaan MUI X Charts

sisälsi kaiken tarvittavan tilastosivun toteuttamiseen. Tekoäly osasi suurimman osan ajasta käyttää kirjaston komponentteja oikein, mutta toisinaan tekoälyn tuottamaan koodiin joutui tekemään korjauksia kirjaston dokumentaatiota hyödyntäen. Projektin ulkopuolella ChatGPT:ltä kysyttäessä tekoälyn kirjastovalintaa ja sen tarjoama käyttöesimerkki oli perinteisessä projektissakin käytetty Recharts.



KUVA 13. Tekoälyprojektin tilastosivu.

### 5.3 Projektien yhteenveto

Yhteenvetona projekteista taulukko 1 esittää projekteissa hyödynnettäväksi valitut kirjastot ja komponentit. Taulukko sisältää myös mahdolliset eri osapuolien perustelut valittujen kirjastojen käytölle.

TAULUKKO 1. Yhteenveto projektien teknologioista

<b>Aihe-alue</b>	<b>Valinta Oma</b>	<b>Perustelu Oma</b>	<b>Valinta Tekoäly</b>	<b>Perustelu Tekoäly</b>
Sovelluskehys, ”pohja”	Next.js	Rajaus	Next.js	Rajaus
Tyylittely	Tailwind CSS	Rajaus	Tailwind CSS	Rajaus
Komponentti- kirjasto	shadcn/ui	Aiempi mielenkiinto, avoimuus / omistajuus, tyylitelty Tailwindillä	Material UI (MUI)	Tunnettu, mukautettava, kattava, laaja yhteisö
Käytetyt komponentit	Button, Card, Navigation Menu, Scroll Area, (Typography)	Tarvittavat komponentit työkaluun sisällön esittämiseen	AppBar, Box, Button, Card, CardContent, CircularPro- gress, Container, Grid, Toolbar, Typography	Tekoälyn käyttämät, tarvittavat komponentit
Visualisointi- kirjasto	Recharts	Tiedonhaun perusteella suosituu valinta React- projekteihin	MUI X Charts	Tyylin johdon- mukaisuus komponenttien kanssa
JSON- validointikirjasto	Ajv	Tiedonhaun perusteella suosituuin	Ajv	Tekoälyn ainoa ehdotus
Raahaa ja pudota -kirjasto	react-dropzone	Tiedonhaun perusteella suosituuin	react-dropzone	Tekoälyn ainoa ehdotus



## 6 TULOSTEN VERTAILU

Molempien projektitoteutuksien lopputulokset olivat toiminnallisuudeltaan sekä rakenteeltaan lopulta melko lähellä toisiaan. Toteutusten sisältö oli oman harkintakyvyn mukaan molemmissa projekteissa hyvän laatuista sekä esimerkiksi jatkokkehityksen (kuten useamman pelin tilastojen yhteenvetodiagrammit) kannalta käyttökelpoista ja hyvin skaalautuvaa. Myöskään projektien kehitysvaiheissa ei ilmennyt merkittävämpiä eroavaisuuksia. Projekteja toteutettaessa tuli kuitenkin esille myös osa-alueita, joissa toteutustavat eroavat toisistaan. Erilaisia hyötyjä havaittiin molemmissa toteutustavoissa.

### 6.1 Projektien toteutusnopeus

Suurin huomattava eroavaisuus näiden kahden vertailtavan ohjelmistoprojektin välillä oli niiden toteutukseen käytetty aikamäärä. Perinteisen ohjelmistoprojektin toteuttamiseen aikaa kului karkeasti noin 24 tuntia. Tekoälyavusteiseen projektiin aikaa kului vain noin 8,5 tuntia. Tekoälyprojektin toteuttaminen oli siis pyöristetysti 65 prosenttia nopeampi verrattuna perinteiseen toimintamalliin.

GitHub Copilot sisällyttää kotisivuillaan väitteen, että Copilot tarjoaa käyttäjilleen 55 prosenttia nopeampaa koodaamista (Copilot n.d.). Tämän työn pohjalta kyseinen väite vaikuttaisi olevan todenmukainen. Tässä yhteydessä tulee huomioida se seikka, että Copilotin väite pohjautuu suurempaan kokonaisuuteen. Tässä työssä toteutetut projektit tarjoavat vain yhden vertailtavan datapisteen, joka vääristää työssä saadun tuloksen tarkkuutta.

Perinteisessä projektissa eniten aikaa vieneet osa-alueet olivat tiedonhaku, käytettävien teknologioiden dokumentaation selaaminen ja referointi, sekä koodauksessa itse kirjoittaminen. Tekoälyn käytöllä voidaan korvata suuri osuus juuri näihin osa-alueisiin kuluneesta ajasta. Generatiivisen koodisyötteen avulla voidaan karsia projektin kirjoittamiseen kuluvaan aikaa, kun kaiken kirjoittamisen sijasta voidaan vain tehdä pienempiä lopullisia muutoksia generoituun sisältöön. Tästä ominaisuudesta voidaan katsoa olevan hyötyä esimerkiksi niille kehittäjille, jotka

ovat hitaampia kirjoittamisessa. Tiedonhaun sekä dokumentaatioiden tarkan läpikäynnin tarve myös vähenee tekoälyavustajan, kuten Copilot Chatin, tarjoaman tietämyksen seurauksena. Tekoäly ei kuitenkaan kokonaan poissulkenut näidenkään osa-alueiden tarvetta.

Tekoälyprojektin kohdalla aikaa kului etenkin käytettävän tekoälyn ohjaamiseen kohti haluttua lopputulosta. Myös tekoälyn tarjoamiin virheellisiin tai puutteellisiin vastauksiin johtaneiden syötteiden uudelleenmuotoiluun ja tarkentamiseen kului huomattavasti aikaa.

## **6.2 Kehitystyökalujen käyttäjäystävällisyys**

Molemmissa projekteissa kehitystyökalujen käyttö tuntui luontevalta. Käytössä olleeseen Visual Studio Code -editoriin (kuva 14) tekoälyprojektissa käytetyt työkalut saatiin helposti lisättyä kahden laajennuksen, GitHub Copilot ja GitHub Copilot Chat, avulla. Kyseiset laajennukset integroivat tekoälyn koko editoriin siistillä ja luontevalla tavalla (muun muassa koodi, terminaali ja versionhallinta). Laajennukset saa myös halutessaan pysäytettyä, jos esimerkiksi ei haluta tekoälyn generoivan kirjoitettavan sisällön perusteella automaattitäydennysehdotuksia. Visual Studio Code on ilman tekoälyäkin hyvin muokattava ja monipuolinen editori kaikkine laajennuksineen.

The screenshot shows the Visual Studio Code interface. On the left, the Copilot chat window is open, displaying a conversation with a user named Sami-Saastamoinen. The chat content includes a question about handling JSON validation and a detailed code modification for the `Home` component. The code modification involves adding a `isValid` state variable, updating the `onDrop` and `onClick` handlers to use `validateJson`, and updating the `Button` component's `disabled` prop. The chat also includes a note about the `validateJson` function's return type and a prompt to ask for more help.

The main code editor shows the `Home` component code, which is a React function component. It uses `useState` for `files` and `isValid`, and `useDropzone` for file uploads. The component renders a `Card` with a `Button` for file selection and a `Link` to the stats page. It also displays validation errors if any.

```

export default function Home() {
  const [files, setFiles] = useState([]);
  const [isValid, setIsValid] = useState(false); // Add this line
  const { getRootProps, getInputProps } = useDropzone({
    accept: [ 'application/json' ],
    maxFiles: 1,
    onDrop: (acceptedFiles) => {
      acceptedFiles.forEach((file) => {
        const reader = new FileReader();
        reader.onabort = () => console.log("file reading was aborted");
        reader.onerror = () => console.log("file reading has failed");
        reader.onload = () => {
          const json = JSON.parse(reader.result);
          setIsValid(validateJson(json)); // Modify this line
        };
        reader.readAsText(file);
      });
      setFiles(acceptedFiles);
    },
  });
  // ...
  <Button
    variant="contained"
    color="primary"
    disabled={files.length === 0 || !isValid} // Modify this line
    onClick={() => {
      const reader = new FileReader();
      reader.onload = function (e) {
        localStorage.setItem("fileContent", e.target.result);
      };
      reader.readAsText(files[0]);
      localStorage.setItem("fileName", files[0].name);
    }}
  />
  <Link href="/stats">See Your stats</Link>
  </Button>
  // ...
}

```

KUVA 14. Visual Studio Coden työympäristö tekoälyprojektissa, sivupalkissa avoinna Copilot Chat.

Koodissa sekä terminaalissa kursorin tai tekstivalinnan kohdalle voidaan oletuspikanäppäimellä "Ctrl+I" avata Copilotin Inline Chat -ominaisuus, jonka avulla kyseiseen kohtaan voidaan pyytää muokkauksia, uutta sisältöä, selitystä tai komentoehdotuksia. Versionhallinnan osalta Copilot tarjoaa pysyvien muutosten viestien (commit message) generoinnin. Copilotin yrityksille tarjoamalla Copilot Enterprise -versiolla Copilot voitaisiin myös integroida GitHub-verkkosivuille, jolloin tarjolla olisi vetopyynnöille (pull request) suunnattuja ominaisuuksia kuten eroanalyysi (diff analysis) sekä kuvaus ja yhteenveto (description and summarization) (Copilot n.d.).

### 6.3 Virheen käsittely

Koodissa ilmenevä virheen käsittely on otettu huomioon kummassakin projektissa, tosin hieman laajemmin tekoälyprojektissa, jonka ehdotukset yleisesti sisältävät jo valmiiksi mahdollisten virhetilanteiden käsittelemisen. Perinteisessä toteutustavassa virheen käsittely voi usein olla manuaalisempi tehtävä, joka toteutetaan toimivaan ominaisuuteen jälkikäteen.

Projekteissa käytetty JavaScript on GitHubin mukaan yksi Copilotin parhaiten tuetuista ohjelmointikielistä, koska se on edustettuna monimuotoisesti ja määrätään laajasti GitHubin julkisissa projekteissa, joita on käytetty Copilotissa koulutusdatana (Copilot n.d.). Tämä voi osaltaan vaikuttaa Copilotin vastausten laatuun sekä vankempaan virheen käsittelyyn hyödyntämiseen sen ehdottamassa koodisisällössä, jotka ainakin tämän työn perusteella pohjautuvat laadukkaaseen koulutusdataan.

### 6.4 Toteutusten virheellisyys

Vaikkakin tekoälyprojektin toteutukseen kului huomattavasti vähemmän aikaa ja käyttökokemukseltaan se oli miellyttävämpi, kyseinen toteutustapa ei ollut kaikilla osa-alueilla suoraan parempi. Perinteisessä projektissa tuotettu sisältö oli suurimman osan ajasta toimivaa toteuttamista, koska kirjoittaminen tehtiin käytettyjen teknologioiden omien dokumentaatioiden sekä esimerkkien pohjalta, jolloin niiden soveltaminen kyseiseen projektiin sopiviksi oli virheettömämpää. Tietysti perinteisessä projektissakin esiintyi myös vikoja, mutta niiden korjaamisessa auttoi jo valmiiksi tehty tiedonhaku ja teknologioiden perehtyminen. Perinteisessä projektissa myös esiintyi parempi kokonaisnäkemys toteutetusta projektista, jolloin sitä voitiin käyttää kontekstina soveltamisessa.

Vaikkakin Copilot Chatille pystyy syötteessä tarjoamaan koko työtilan kontekstina, olivat sen tarjoamat vastaukset harvoin ensimmäisellä kysymiskerralla täysin virheettömiä tai toimivia. Usein tekoälyltä joutui kysymään uudelleen haluttua toiminnallisuutta uudelleenmuotoiduilla tai enemmän yksityiskohtia sisältävillä

syötteillä tai tarjoamalla jatkosyötteitä, jonka perusteella tekoälyn haluttiin muokata vastaustaan. Joissain tilanteissa tekoälylle pystyttiin edeltävän perinteisen projektin ja oman aiemman tietämyksen pohjalta tarjoamaan sitä oikeaan suuntaan ohjaavia syötteitä, kuten ”eikö asia x voisi olla näin” tai ”mielestäni x”. Useimmissa vastaavissa tilanteissa tekoäly katsoi syötetyn asian oikeaksi ja sai toteutettua tarvittavat muutokset, mutta joka kerta tämäkään ei tuottanut haluttua tulosta.

Myös Stack Overflow:n tuottamassa kyselyssä noin joka neljäs vastaaja kertoo tekoälyn vastauksiin luottamisen olevan haaste tekoälytyökalujen käytössä. Lähes saman verran vastaajista kokee ongelmaksi tekoälyn kyvyttömyyden ratkaista korkeamman monimutkaisuuden ongelmia. (Stack Overflow 2024.)

Vertailun perusteella voidaan päätellä, että tekoälytyökalut eivät ole vielä sillä tasolla, että ne voisivat luoda laajaa ja luotettavaa projektimuotoista sisältöä itsenäisesti. Nämä työkalut tarvitsevat edelleen ihmisen syötettä toimiakseen, joka onkin niiden tämänhetkinen tarkoitus.

## 6.5 Projekteista oppiminen

Molemmista projekteista löytyi seikkoja, joiden pohjalta oppimalla voi kehittää tiettyjä taitoja tullakseen paremmaksi ohjelmistokehittäjäksi. Perinteisen projektin toteuttamisessa oppii löytämään tehokkaasti haluttua tietoa sekä toteuttamaan soveltavaa sisältöä dokumentaation pohjalta, kun käsittelyssä on uusia kirjastoja tai teknologioita. Myös koodissa esiintyvät, mahdollisesti hieman toistuvat rakenteet ja sisällöt voivat auttaa kehittäjää painamaan mieleensä käytettävät asiat paremmin seuraavia teknologioiden käyttökertoja varten.

Tekoälynkin hyödyntäminen voi osaltaan vauhdittaa ja sujuvoittaa esimerkiksi uusien teknologioiden oppimista, kunhan näiden työkalujen käyttäjä vain pitää myös tarpeelliset dokumentaatiot lähettyvillä. Myös kriittisyyden sekä oman harkintakyvyn harjoittaminen ovat tärkeitä oppimisaiheita työskennellessä tekoälyä vastauksia ohjaavasti näiden työkalujen kanssa.

## 7 POHDINTA

Tämän työn löydösten pohjalta tekoälytyökalujen käytöllä ei tilanteen mukaan vältty omatoimiselta tiedonhaulta, teknologioihin perehtymiseltä tai manuaaliselta toteutusten korjaamiselta ja muokkaamiselta. Tekoälyyn ei myöskään voi työn perusteella sokeasti luottaa, sillä se sisälsi useita virheitä. Tekoälyn tuottaman sisällön validoinnissa auttoi se, että perinteisen toteutustavan projekti oli suoritettu ennen tekoälyprojektia, jolloin tekoälyn virheitä huomasi helpommin. Tilanne voisi olla kuitenkin eri, jos aiempaa kokemusta tai tuntemusta olisi vähemmän, jolloin virheitä ei välttämättä huomaa heti ja tällöin korjaamiseen jälkikäteen voi joutua näkemään enemmän vaivaa.

Tekoälytyökalujen käytössä ovat kuitenkin myös hyvät puolensa. Projektien nopeampi ja tehokkaampi toteuttaminen sekä työkalujen käyttömieluisuus kohottavat osaltaan myös kehittäjien motivaatiota ja mielenkiintoa projekteja ja työskentelyä kohtaan. Tämä ohjelmistokehittäjien tyytyväisyys entisestään kasvattaa toteuttamisen tehokkuutta ja tuotteliaisuutta, mistä hyötyvät myös muut asianmukaiset tahot. GitHubin mukaan joka kolmas Fortune 500 -yrityksistä hyödyntää GitHub Copilotin työkaluja, joten niiden käytöstä luulisi olevan jotain hyötyäkin (Copilot n.d.).

Tämän takia tekoälytyökaluja ei tulisiakaan pitää ohjelmistokehittäjien korvaajina, kuten nykypäivänä puolittotisesti päädytään ajattelemaan, vaan juuri nimensä mukaisesti avustavina työkaluina, joiden avulla projekteja ja ohjelmistokehitystä pysytään nopeuttamaan ja helpottamaan. Joten ainakaan lähitulevaisuudessa ei tarvitse vielä murehtia, että ohjelmistokehittäjien työpaikkoja vähenisi merkittävässä määrin tekoälytyökalujen takia. Jonkunhan pitää myös kyseiset tekoälyt sekä niiden sovellukset kehittää.

Jatkon kannalta työstä kerättyjä tuloksia voitaisiin parantaa esimerkiksi laajentamalla projektien skaalaa tai toteuttamalla projekteja eri alustoilla ja ohjelmointikielillä. Tämä myös kasvattaisi vertailukelpoisten projektien lukumäärää, jolloin niistä muodostuisi suurempi tietojoukko. Tällöin generatiivisten tekoälytyökalujen käytön hyödyistä ja kehityksestä saataisiin tarkempi kuva lähempänä keskiarvoa.

## LÄHTEET

Altmann, Y. 2023. What is AI? Development and History of Artificial Intelligence. Verkkoartikkeli. Viitattu 15.4.2024. <https://omq.ai/blog/history-of-ai/>

Azure. n.d. What is computer vision? Verkkosivu. Viitattu 29.4.2024. <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-computer-vision#object-identification>

Berliner, H. 1980. Computer Backgammon. Scientific American 242 (6), 64-72. Viitattu 17.4.2024. <https://www.bkgm.com/articles/Berliner/ComputerBackgammon/index.html>

Copilot. n.d. GitHub. Verkkosivu. Viitattu 16.5.2024. <https://github.com/features/copilot>

Coreggio, G. 2023. EU AI Act Approved: Everything You Need to Know on the Artificial Intelligence legislation in Europe. Verkkoartikkeli. Viitattu 26.4.2024. <https://www.linkedin.com/pulse/eu-ai-act-approved-everything-you-need-know-europe-giulio-coraggio-lpw3f/>

Coursera. 2024. What Is Machine Learning? Definition, Types, and Examples. Verkkoartikkeli. Viitattu 30.4.2024. <https://www.coursera.org/articles/what-is-machine-learning>

ECMA-404. 2017. The JSON Data Interchange Syntax. Standardi. 2. painos. Viitattu 5.6.2024. [https://ecma-international.org/wp-content/uploads/ECMA-404\\_2nd\\_edition\\_december\\_2017.pdf](https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf)

Google DeepMind. n.d. AlphaGo. Verkkosivu. Viitattu 17.4.2024. <https://deepmind.google/technologies/alphago/>

Grobelnik, M., Perset, K. & Russell, S. 2024. What is AI? Can you make a clear distinction between AI and non-AI systems? Verkkoartikkeli. Viitattu 26.4.2024. <https://oecd.ai/en/wonk/definition>

Haenlein, M. & Kaplan, A. 2019. A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. California management review 61 (4), 5—14. Viitattu 13.4.2024. <https://doi.org/10.1177/0008125619864925>

Herbert, D. 2023. What is React.js? Uses, Examples, & More. Verkkoartikkeli. Viitattu 11.4.2024. <https://blog.hubspot.com/website/react-js>

IBM. n.d.a. What Is machine learning (ML)? Verkkosivu. Viitattu 30.4.2024. <https://www.ibm.com/topics/machine-learning>

IBM. n.d.b. What is natural language processing (NLP)? Verkkosivu. Viitattu 19.5.2024. <https://www.ibm.com/topics/natural-language-processing>

Introduction. n.d. Next.js. Verkkosivu. Viitattu 12.4.2024. <https://nextjs.org/docs>

- Lu, A. 2022. Four Seasons of AI. The brief history of Artificial Intelligence (I). Verkkoartikkeli. Viitattu 13.4.2024. <https://www.labelf.ai/blog/four-seasons-of-ai-the-brief-history-of-artificial-intelligence-i>
- Lutkevich, B. 2022. AI Winter. Verkkoartikkeli. Viitattu 17.4.2024. <https://www.techtarget.com/searchenterpriseai/definition/AI-winter>
- McCarthy, J. 2007. WHAT IS ARTIFICIAL INTELLIGENCE? PDF-artikkeli. Viitattu 26.4.2024. <http://www-formal.stanford.edu/jmc/whatisai.pdf>
- McCarthy, J., Minsky, M. L., Rochester, N. & Shannon, C. E. 1955. A Proposal for the Dartmouth Summer Research Project of Artificial Intelligence. PDF-artikkeli. Viitattu 25.4.2024. <http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf>
- Meta. 2024. Introducing Meta Llama 3: The most capable openly available LLM to date. Verkkoartikkeli. Viitattu 20.5.2024. <https://ai.meta.com/blog/meta-llama-3/>
- Next.js. n.d. The React Framework for the Web. Verkkosivu. Viitattu 12.4.2024. <https://nextjs.org>
- OpenAI. 2024. Hello GPT-4o. Verkkoartikkeli. Viitattu 14.5.2024. <https://openai.com/index/hello-gpt-4o/>
- OpenAI. n.d. The fastest and most powerful platform for building AI products. Verkkosivu. Viitattu 14.5.2024. <https://openai.com/api/>
- Ozdemir, S. 2023. Quick Start Guide to Large Language Models: Strategies and Best Practices for Using ChatGPT and Other LLMs. E-kirja. Hoboken, New Jersey: Addison-Wesley Professional. Viitattu 20.5.2024. <https://learning.oreilly.com/library/view/quick-start-guide/9780138199425/>
- Pricing. n.d. OpenAI. Verkkosivu. Viitattu 14.5.2024. <https://openai.com/chat-gpt/pricing/>
- Rathod, J. 2020. Branches in Artificial Intelligence to Transform Your Businesses! Verkkoartikkeli. Viitattu 29.4.2024. <https://jashrathod.github.io/2020-11-24-branches-in-artificial-intelligence-to-transform-your-businesses/>
- React. n.d. The library for web and native user interfaces. Verkkosivu. Viitattu 11.4.2024. <https://react.dev/>
- React Native. n.d. Learn once, write anywhere. Verkkosivu. Viitattu 11.4.2024. <https://reactnative.dev/>
- Start a New React Project. n.d. React. Verkkosivu. Viitattu 11.4.2024. <https://react.dev/learn/start-a-new-react-project>
- Stack Overflow. 2024. Developers get by with a little help from AI: Stack Overflow Knows code assistant pulse survey results. Verkkoartikkeli. Viitattu



5.6.2024. <https://stackoverflow.blog/2024/05/29/developers-get-by-with-a-little-help-from-ai-stack-overflow-knows-code-assistant-pulse-survey-results/>

Stollnitz, B. 2023. The Transformer architecture of GPT models. Verkkoartikkeli. Viitattu 20.5.2024. <https://bea.stollnitz.com/blog/gpt-transformer/>

Tailwind. n.d. Rapidly build modern websites without ever leaving your HTML. Verkkosivu. Viitattu 5.6.2024. <https://tailwindcss.com>

Umbrello, S. 2021. AI Winter. Teoksessa Frana, P. & Klein, M. (toim.) Encyclopedia of Artificial Intelligence : The Past, Present, and Future of AI. E-kirja. Santa Barbara, California: ABC-CLIO, 7—8.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L. & Polosukhin, I. 2017. Attention Is All You Need. PDF-tutkimusartikkeli. Viitattu 5.6.2024. <https://doi.org/10.48550/arXiv.1706.03762>

Wang, L., Liu, Z., Liu, A. & Tao, F. 2021. Artificial intelligence in product lifecycle management. The International Journal of Advanced Manufacturing Technology 114 (1), 771—796. Viitattu 15.4.2024. <https://doi.org/10.1007/s00170-021-06882-1>

Weizenbaum, J. 1966. ELIZA—a computer program for the study of natural language communication between man and machine. Communications of the ACM 9 (1), 36—45. Viitattu 16.4.2024. <https://doi.org/10.1145/365153.365168>

Wikipedia. n.d. ELIZA. Verkkoartikkeli. Viitattu 16.4.2024. <https://en.wikipedia.org/wiki/ELIZA>