

Markus Palomäki

# KIELIMALLIT KOODARIN APUNA OSANA JATKUVAA PIENKEHITYSTÄ

Diplomityö  
Informaatioteknologian ja viestinnän tiedekunta  
Professori Kari Systä  
Professori Pekka Abrahamsson  
Toukokuu 2024

# TIIVISTELMÄ

Markus Palomäki: Kielimallit koodarin apuna osana jatkuvaa pienkehitystä  
Diplomityö  
Tampereen yliopisto  
Tietotekniikan DI-ohjelma  
Toukokuu 2024

---

Tietokonepelaaja shakkipeleissä, esineiden tunnistaminen valokuvista, robotti-imurit. Näiden ja lukemattomien muiden työkalujen tai palvelujen taustalla toimivat tekoälyt ovat olleet useiden vuosien ajan osa arkielämäämme. Viimeisten vuosien aikana tutkimukset tekoälystä ovat ottaneet suuria harppauksia tuoden yksinkertaisten, yhteen asiaan erikoistuneiden tekoälyjen rinnalle suuria kielimalleja hyödyntäviä, opetettavia tekoälyjä, joiden opettamiseen on käytetty suuria määriä tietoa.

Tämän diplomityön tarkoituksena on tehdä sukellus tämän päivän suuriin kielimalleihin (LLM, Large Language Models), sekä perehtyä niiden hyödyntämiseen sovelluskehittäjän näkökulmasta. Tutkimuksessa tarkasteltiin helposti saatavilla olevien, kielimalleja käyttävien sovellusten ja palveluiden hyödyntämistä osana jatkuvaa IT-yrityksissä toteutettavaa pienkehitystä. Tyypillisesti pienkehitystiimien sovelluskehittäjät työskentelevät samanaikaisesti monien projektien ja siten lukuisien eri teknologioiden ja pilvipalvelujen parissa. Kielimalleista voisi olla suurta hyötyä kehittäjille.

Työtä varten toteutettiin kaksi haastetta, joiden ratkaisemiseksi käytettiin kolmea palvelua: GitHub Copilot, Google Bard ja OpenAI ChatGPT. Haasteet olivat kooltaan ja tavoitteiltaan erilaiset: kuvalle CSS-animaation luominen valmiiseen React-komponenttiin on toteutettavissa muuttaman rivin muutoksella, kun taas testiympäristön ja testien luominen Contentful-sisällönhallintajärjestelmää hyödyntävälle React-verkkosovellukselle vaatii monien uusien kirjasto-jen asentamista, projektitiedostojen muokkaamista ja testien luomista.

Haasteita pyrittiin ratkaisemaan seuraamalla kielimallien antamia ohjeita. Kielimallien välillä aloitussyötteet pidettiin samana ja syötteiden ja vastauksien määrää rajattiin kolmeen. Tällä haluttiin korostaa kielimalleja hyödyntävien sovellusten eroavaisuuksia. Haasteiden lisäksi työssä vertailtiin palvelujen toiminnallisuuksia, tiedon keräämistä ja hinnoittelua, mikä antaa lisätietoa palvelujen hyödynnettävyydestä IT-yrityksissä.

Kokeellisessa tutkimuksessa havaittiin, että kolme valittua kielimallisovellusta eivät vastausiltaan suuremmin eronneet toisistaan. Jokainen malleista antoi samankaltaisia vastauksia auttaen kehittäjää etenemään haasteiden toteuttamisessa. Pienin eroin Copilot nousi esille sen käyttämän kielimallin vuoksi, jota on opetettu ja hienosäädetty sovelluskehittäjiä varten, kun taas Bard ja ChatGPT nousivat esille yleisempään käyttöön soveltuvina kielimalleina. Työn aikana Bard oli kehitysvaiheessa, mikä saattoi vaikuttaa sovelluksen valintaan olla auttamatta kehittäjää luomaan animaatiota uskoen kehittäjän pyytävän kuvanluomista. Pienikokoisen muutoksen, tyylianimaation luomisen toteuttaminen, onnistui kolmen vastauksen ja kehittäjän pienien korjausten jälkeen. Kehitysympäristön luomisessa kolmen vastauksen rajaus ei riittänyt auttamaan kehittäjää luomaan testiympäristöä.

Avainsanat: suuret kielimallit, pienkehitys, tekoäly, OpenAI ChatGPT, Google Bard, Gemini, GitHub Copilot

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality Check -ohjelmalla.

# ABSTRACT

Markus Palomäki: Language Models as Coder's Aid in Continuous Small-Scale Development  
Master of Science Thesis  
Tampere University  
Master's Programme in Information Technology  
May 2024

---

Computer gamers in chess games, object recognition from photographs, robot vacuum cleaners. The artificial intelligences that operate behind these and countless other tools or services have been part of our daily lives for several years. In recent years, research on artificial intelligence has made significant leaps, bringing alongside simple, specialized AIs those utilizing large language models (LLMs), which are teachable and have been trained with vast amounts of data.

The purpose of this thesis is to delve into today's large language models and explore their use from the perspective of an application developer. The study examined the utilization of readily available applications and services using language models as part of continuous small-scale development in IT companies. Typically, developers in small development teams work simultaneously on many projects and thus with numerous different technologies and cloud services. Language models could be of great benefit to developers.

For this work, two challenges were undertaken using three services: GitHub Copilot, Google Bard, and OpenAI ChatGPT. The challenges varied in size and objectives: creating a CSS animation for an image in a ready-made React component can be accomplished with just a few lines of change, whereas creating a testing environment and tests for a React web application using the Contentful content management system requires installing many new libraries, modifying project files, and creating tests.

The challenges were tackled by following the instructions given by the language models. The initial inputs were kept the same across the models, and the number of inputs and responses was limited to three. This was done to highlight the differences between applications utilizing language models. In addition to the challenges, the study compared the functionalities, data gathering, and pricing of the services, providing further information on the usability of the services in IT companies.

The experimental study found that the three selected language model applications did not significantly differ in their responses. Each model provided similar answers, helping the developer to progress in implementing the challenges. With minor differences, Copilot stood out due to its language model, which has been taught and fine-tuned specifically for developers, whereas Bard and ChatGPT are suited for more general use. During the work, Bard was in the development phase, which might have influenced the application's choice not to help the developer in creating animation, believing the developer was asking for image creation. The implementation of a minor change, creating a style animation, was successful after three responses and minor corrections by the developer. The three-response limit was not sufficient to help the developer create a testing environment.

Keywords: large language models, small-scale development, AI, OpenAI ChatGPT, Google Bard, Gemini, GitHub Copilot

The originality of this thesis has been checked using the Turnitin Originality Check service.

# ALKUSANAT

Haluan kiittää kaikkia tukijoita ja tsemppaajia matkan varrella, joiden kannustuksien ja tuen avulla olen saanut lisää puhtia puskea työtä eteenpäin. Kiitos työnantajalle, työyhteisölle ja työkavereille tuesta sekä mahdollisuudesta tehdä diplomityötä työn ohella. Kiitos ohjaajalleni professori Kari Syställe ideoinneista, ohjauksesta ja tsemppaamisesta. Kiitos ystäville ja perheelle tuesta ja kannustuksesta. Ilman teidän kaikkien tukea työn maaliin saaminen olisi ollut huomattavasti vaikeampaa.

Tampereella, 9.5.2024

Markus Palomäki

# SISÄLLYSLUETTELO

1. JOHDANTO .....	1
1.1 Tausta .....	1
1.2 Työn tavoite ja tutkimuskysymys.....	1
2. TEKOÄLY .....	3
2.1 Määritelmä.....	4
2.2 Tekoälyn osa-alueet .....	5
2.3 Neuroverkot.....	6
3. KIELIMALLIT .....	8
3.1 Transformer-arkkitehtuuri.....	9
3.2 GPT .....	12
3.3 PaLM.....	14
3.4 LaMDA .....	15
3.5 OpenAI Codex .....	16
4. TEKOÄLYT JA KIELIMALLIT PALVELUINA .....	17
4.1 ChatGPT .....	18
4.2 Google Bard .....	19
4.3 GitHub Copilot .....	20
4.4 Hinnoittelu ja ominaisuudet.....	21
4.4.1 OpenAI ChatGPT.....	21
4.4.2 Google Bard.....	23
4.4.3 GitHub Copilot .....	23
4.5 Tietoturva, data ja oikeudet.....	25
5. JATKUVA SOVELLUSKEHITTÄMINEN JA PIENKEHITYS .....	27
6. KIELIMALLIEN HYÖDYNTÄMINEN OSANA PIENKEHITYSTÄ.....	28
6.1 Ratkaistavat tehtävät .....	28
6.2 Haaste 1: Tyylianimaation luominen .....	30
6.3 Haaste 2: Testiympäristön ja testien luominen.....	30
6.4 Ratkaisujen toteuttaminen kielimalleja hyödyntämällä .....	31
6.4.1 Haaste 1: Tyylianimaation toteuttaminen .....	31
6.4.2 Haaste 2: Testiympäristön ja testien luominen .....	37
6.5 Kielimallien tulosten vertailu.....	44
6.5.1 Haaste 1: Tyylianimaation luominen.....	44
6.5.2 Haaste 2: Testiympäristön ja testien luominen .....	47
7. TULOKSET .....	50
8. POHDINTA .....	54
LÄHTEET .....	56
LIITE A: TYYLIANIMAATION LUOMISEN ALOITUSSYÖTE (HAASTE 1) .....	64

LIITE B: CHATGPT-KESKUSTELU TYYLIANIMAATION LUOMISEEN .....	66
LIITE C: GITHUB COPILOT -KESKUSTELU TYYLIANIMAATION LUOMISEEN ...	69
LIITE D: CHATGPT-KESKUSTELU TESTIYMPÄRISTÖN JA TESTIEN LUOMISEEN 72	
LIITE E: GOOGLE BARD -KESKUSTELU TESTIYMPÄRISTÖN JA TESTIEN LUOMISEEN .....	78
LIITE F: GITHUB COPILOT -KESKUSTELU TESTIYMPÄRISTÖN JA TESTIEN LUOMISEEN .....	84

## KUVALUETTELO

<b>Kuva 1.</b>	<i>Singulariteetti-hypoteesia kuvaava graafi. Hypoteettinen tulevaisuudenkuva alkaa hetkestä, jolloin ihmisten älykkyys ja tekoäly risteävät. Leikkauspistettä kuvataan kuvassa termillä ”the Singularity”, eli Singulariteetti. [17]</i> .....	5
<b>Kuva 2.</b>	<i>Transformer-arkkitehtuuria kuvaava diagrammi. Kuvakaappaus. [42]</i> .....	10
<b>Kuva 3.</b>	<i>Suurten modernien kielimallien kehittymistä kuvaava sukupuu. Pystyjana kuvaa tunnettujen kielimallien julkaisuvuoden. Kielimallien taustaelementit kertovat koodin avoimuudesta (taustaväri avointa lähdekoodia, valkoinen tausta reunaviivalla suljettua koodia). Värilliset (muut kuin harmaat) kuvaavat Transformer-arkkitehtuurin kielimalleja. Oikealla alareunassa pylväsdiagrammi kuvaa eri yritysten ja instituutioiden julkaisemien kielimallien lukumäärää sekä niiden avoimuutta. [52]</i> .....	12
<b>Kuva 4.</b>	<i>ChatGPT:n käyttöliittymä. Kuvakaappaus.</i> .....	18
<b>Kuva 5.</b>	<i>Nykyisen Google Geminin (ennen Google Bard) käyttöliittymä. Gemini-päivitys ei tuonut ulkoasuun merkittäviä muutoksia. Kuvakaappaus.</i> .....	19
<b>Kuva 6.</b>	<i>Visual Studio Code käyttöliittymä. Oikeassa reunassa paneeli GitHub Copilot keskustelulle. Muokattu kuvakaappaus.</i> .....	20
<b>Kuva 7.</b>	<i>Ruudunkaappaus toteutetusta verkkosovelluksesta. Testiä varten luotu yksinkertainen käyttöliittymä hyödynsi React.js JavaScript-kehystä sekä Contentful-sisällönhallintajärjestelmää. Kuvakaappaus.</i> .....	29
<b>Kuva 8.</b>	<i>ChatGPT:n ensimmäisen vastauksen avulla saatiin toimiva animaatio, mutta se sulautti kuvan päällä olleen tekstin sivun taustan valkoisen värin vuoksi. Kuvassa tekstit valittuna niiden esiin saamiseksi. Kuvakaappaus.</i> .....	32
<b>Kuva 9.</b>	<i>ChatGPT:n kolmannen vastauksen jälkeen animaatio aktivoituu hiirestä, mutta kuvasuhde vääristyy. Kuvakaappaus.</i> .....	33
<b>Kuva 10.</b>	<i>Copilot -keskustelusta saatu ensimmäinen vastaus toteutti halutun toiminnallisuuden, mutta vääristyneellä kuvasuhteella. Kuvakaappaus.</i> .....	35
<b>Kuva 11.</b>	<i>Copilotin antaman toisen vastauksen lopputulos, mikä suurentaa kuvan sijaan koko ylemmän tason elementtiä. Kuvakaappaus.</i> .....	36
<b>Kuva 12.</b>	<i>Google Bard ei ymmärtänyt aloitusyötettä oikein. Kuvakaappaus.</i> .....	45

## LYHENTEET JA MERKINNÄT

AI	engl. Artificial Intelligence, tekoäly tai keinoäly
API	engl. Application Programming Interface, ohjelmointirajapinta
BERT	engl. Bibirectional Encoder Representations from Transformers, Googlen kehittämä kielimalli
ChatGPT	OpenAI:n tarjoama, tekstipohjainen verkkosovellus kielimallin käyttämiseen
CMS	engl. Content Management System, sisällönhallintajärjestelmä, minkä avulla verkkosovelluksen sisältöä voidaan päivittää erillisen käyttöliittymän kautta
DL	engl. Deep Learning, syväoppiminen
et al.	lat. et alii, tai et aliae, ja muut
GPT	Generative Pre-training Transformer, OpenAI:n kehittämä kielimalli
GPTs	OpenAI:n tarjoama, ChatGPT-keskustelun päälle rakennettu ohjelma, joka on räätälöity tiettyä tarkoitusta varten
InstructGPT	OpenAI:n julkaisema kielimalli, mikä perustuu GPT-3 versioon
LaMDA	engl. Language Model for Dialogue Applications, Googlen kehittämä kielimalli
ML	engl. Machine Learning, koneoppiminen
NLP	engl. Natural Language Processing, luonnollisen kielen käsittely
PaLM	engl. Pathways Language Model
RLHF	engl. Reinforcement Learning from Human Feedback, vahvistavaa oppimista ihmispalautteesta

# 1. JOHDANTO

## 1.1 Tausta

Vuoden 2023 alussa ChatGPT nousi sosiaalisessa mediassa puheenaiheeksi tekoälynä, joka pystyi vastaamaan käyttäjien kysymyksiin ja pyyntöihin uskottavasti. Joulukuussa 2015 perustetun OpenAI:n julkaisemaa chattibottia ja sen tuotoksia jaettiin sosiaalisessa mediassa [1], ja pian siitä uutisoitiin muun muassa hakukoneyhtiön Googlen uhkana [2]. Kielimallien ja niitä hyödyntävien sovelluksien sekä työkalujen yleistyessä ja hakiessaan yritykset tunnistivat niihin liittyvän suuren potentiaalin mieltivät niiden mahdollista suurta potentiaalia hyödyntää työtehtävien rinnalla apuvälineinä kuten ongelmatilanteiden ratkomisessa ja ideoinnissa hyödyntämisessä. Muun muassa Science-tiedelehden julkaisussa Shakked Noy:n ja Whitney Zhangin toteuttamassa tutkimuksessa havaittiin ChatGPT-sovelluksen käytön nostaneen tuottavuutta 40% sekä laatua 18% [3].

IT-yrityksissä tapahtuvassa jatkuvassa pienkehityksessä projektia jatkokehitetään usein ominaisuuksin sekä ylläpidetään korjaten ilmenneitä ongelmia. Pienkehitys on palvelun jatkokehittämistä, jossa jokaiselle kuukaudelle on varattu sovittu määrä työskentelyaikaa. Sen koko riippuu osapuolien tekemästä sopimuksesta, mutta tyypillistä on, että kehitystyö ei ole kokopäiväistä, vaan määränä voi olla esimerkiksi viisi työpäivää kuukaudessa.

Pienkehityksessä toimivalla sovelluskehittäjällä voi olla samanaikaisesti useita projekteja, joissa käytetyt tekniikat ja pilviympäristöt voivat erota suuresti. Tällaisessa työtehtävässä kehittäjällä pitää olla tietoa ja ymmärrystä monista eri tekniikoista ja valmius erilaisten ongelmien ratkomiseen. Lisäksi sovelluskehittäjällä käytettävä aika on rajallista, mikä voi lisätä kiireellisyyttä, priorisointia ja haastaa projektien samanaikaista hallintaa. Tällaisessa kuormittavassa työssä kielimalleilta voisi mahdollisesti saada tukea.

## 1.2 Työn tavoite ja tutkimuskysymys

Diplomityössä tutustuttiin eri kielimalleja tarjoaviin yrityksiin ja niiden tuotteisiin sekä niiden hyödyntämiseen tehtävien ja ongelmien ratkaisussa. Työkalujen valinta rajattiin työkaluihin ja palveluihin, jotka pohjautuvat verkkopalveluihin ja täten ovat helposti jokaisen saatavilla. Tämän avulla palveluiden käyttö ei vaadi käyttäjältä tehokkaita laitteistoja tai tallennustilaa, jotta raskaita kielimalleja pystytään suorittamaan paikallisesti kehittäjän

laitteilla. Tarkasteltaviksi palveluiksi valittiin Google Bard, GitHub Copilot sekä OpenAI ChatGPT.

Vertailevaan tutkimukseen luodut kaksi haastetta ovat kooltaan ja vaikeusasteiltaan erilaiset. Ensimmäisessä haasteessa yritetään luoda kielimalleja hyödyntäen React-komponentissa määritetylle kuvalle CSS-animaatio, jossa elementin ylle viety hiiri käynnistää kuvaa suurentavan animaation. Toisessa haasteessa yritetään luoda testiympäristö ja testejä React-kirjastoa ja Contentful-sisällönhallintaa hyödyntävälle verkkosovellukselle. Samalla tutkitaan palveluiden kuvauksia vertailemalla niiden lupausta tiedon keräämisestä ja kerätyn tiedon hyödyntämisestä, palveluiden tarjoamista ominaisuuksista sekä hinnoista.

Seuraavassa luvussa perehdytään käsitteeseen tekoäly, sen historiaan sekä sen eri osa-alueisiin. Luvussa kolme perehdytään kielimalleja hyödyntävään Transformer-arkkitehtuuriin ja eri kielimalliperheisiin. Neljännessä luvussa tutustutaan valittuihin palveluihin, niiden nopeaan kehittymiseen viimeisien lähivuosien aikana, hinnoitteluihin ja käyttäjän tiedon käyttöön. Viidennessä luvussa käsitellään pienkehitystä. Kuudennessa ja seitsemännessä luvussa kerrotaan luoduista haasteista, niiden ratkaisemisesta kielimalli sovelluksia hyödyntäen ja tulosten vertaamisesta. Kahdeksas luku esittelee tulosten pohdinnan.

Työssä kerrotaan kielimalleista sen hetkisten saatavilla olevien tietojen perusteella. Moni malleista ja palveluista saivat diplomityön toteuttamisen ja kirjoittamisen varrella päivityksiä, mikä kertoo alan nopeasta kehittymisestä. Tästä yhtenä esimerkkinä voidaan nostaa Google Bard -keskustelusovellus, jota päivitettiin käyttämään eri kielimallia sekä myöhemmin uudelleen brändättiin Google Geminiksi.

## 2. TEKOÄLY

Monelle henkilölle sana ”tekoäly” saattaa olla uusi tuttavuus, vaikka monissa eri päivittäin käytettävissä palveluissa jo hyödynnetään tekoälyjä. Haet sitten navigaattorilla parasta reittiä, luet sinulle nostettuja kiinnostavia artikkeleita uutissovelluksessa tai valitset illaksi elokuvan katsottavaksi, palvelujen taustalla voi toimia tekoäly, jonka avulla sinulle ehdotetaan parasta reittiä tai elokuvaa. Akateemisessa maailmassa tekoäly on vakiintunut käsite, sillä sitä on tutkittu jo useiden vuosikymmenien ajan.

Sanan ”tekoäly” historia ulottuu 1900-luvun puolivälille. Sen englannin kielisen termin ”artificial intelligence” loi John McCarthy vuonna 1956, jolloin se esiteltiin uutena tutkimusalana hänen sekä Marvin Minskyn järjestämässä seminaarissa ”Dartmouth Summer Research Project on Artificial Intelligence” [4–6]. Vaikka tekoälyn termi keksittiin tuolloin, Alan Turing kuvaili jo vuonna 1935 abstraktitasolla ”Turing’s stored-program”-laskentakonekonseptia, joka kykeni lukemaan ja kirjoittamaan symboleja rajattoman muistin ja sitä edestakaisin liikkuvalla lukijalla [7].

Nykyisin universaalina Turingin koneena (engl. ”Universal Turing Machine”) tunnetulla konseptilla esitettiin ideaa ohjelmalle, joka kykenee muokkaamaan ja parantamaan itseään. Myöhemmin vuonna 1950 Turing kehitti teoreettisen mallin tietokoneen testaamiseen, selvittääkseen, pystyykö tietokone ajattelemaan, reagoimaan ja keskustelemaan kuten tunteva olio. Tämä tunnetaan Turingin testinä (engl. ”Turing Test”) [7,8]. Turingin testissä kuulustelija yrittää kertoa, kumpi itseään ihmiseksi väittävistä ihmisestä sekä tekoälystä on oikeasti ihminen tai tekoäly. Tekoälyn harhauttaessa kuulustelijan tekoälyn voidaan kertoa läpäisseen testin.

Ensimmäisenä tekoälysovelluksena voidaan pitää Allen Newellin et al. luomaa lauseen todistamisen ohjelmaa vuodelta 1956 [7,9]. Tämä esitettiin aiemmassa kappaleessa mainitussa McCarthy ja Minskyn järjestämässä seminaarissa. Newell et al. kertoo heidän julkaisemassaan artikkelissaan [10], että heidän loogisen teorian ohjelma (engl. ”Logic theory machine”) pohjautuu samankaltaisiin heuristisiin menetelmiin<sup>1</sup>, joita voidaan havaita ihmisten ongelmanratkaisemisen yhteydessä. Lisäksi he muun muassa hyödynsivät symboleja tiedon esittämiseen ja algoritmeja loogisten ongelmien ratkaisemiseen. Nämä ideat ovat vaikuttaneet tekoälyn sekä myöhemmin kielimallien kehityksessä.

---

<sup>1</sup> Heuristinen menetelmä eli ongelmanratkaisumenetelmä. Menetelmän avulla pyritään pääsemään nopeasti mahdollisimman lähelle lopputulosta. Nämä ovat kokeiluun ja arvaukseen perustuvia menetelmiä [11].

## 2.1 Määritelmä

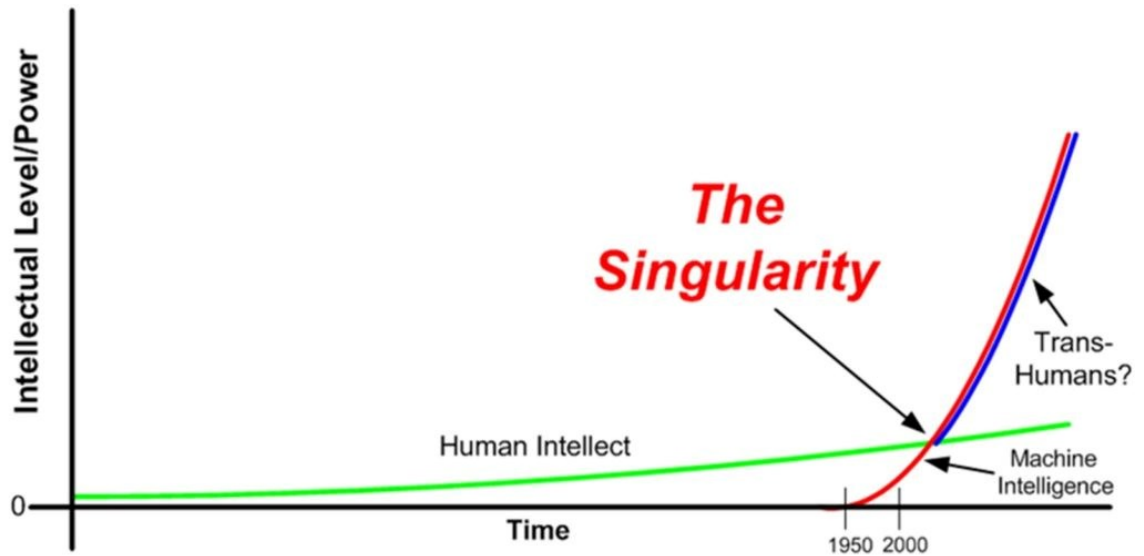
Tekoäly – josta voidaan käyttää myös nimeä keinoäly – on tietokoneen suorittama ohjelma tai järjestelmä, jonka suorittamiseen tarvitaan älykkyyttä [5]. Vaikka yleisesti tietynlaisia laskentaproseduureja ei kutsuta älykkäiksi, voidaan älykkyydellä viitata laskennalliseen osaan erilaisten päämäärien saavuttamisessa [12].

Kuvata ja selittää termi ”tekoäly” on lähestytty ihmiskeskeisen ja rationalistisen lähestymisen kautta. Sen kuvaamiseen ja tutkimiseen voidaan käyttää neljää eri kategoriaa: inhimillisesti ja rationaalisesti ajattelevat sekä toimivat kategoriat [13]. *Inhimillisellä ajattelun* tasolla tekoälyä tutkitaan ihmisen kaltaisena älyn omaavana laitteena, joka kykenee aktiviteetteihin kuten oppimaan ja ongelmien ratkaisemiseen. *Rationaalisella ajattelun* tasolla tekoäly on tutkimusala tutkia henkisiä, havainnointi, perustelu ja toimintakykyjä laskennallisten mallien avulla. *Inhimillisen toiminnan tasolla* tekoäly käsittää tietokoneiden tutkimista kuten sitä, kuinka ne pystyisivät suoriutumaan tehtävistä tehokkaammin kuin ihmiset, kun taas *rationaalisen toiminnan tasolla* tutkimus keskittyy älykkäiden agenttien suunnitteluun.

Älykkäitä järjestelmiä, jotka suorittavat yksinkertaisempaa määriteltyä tehtävää kuten esineiden tunnistaminen kuvista tai shakin pelaaminen ihmispelaajaa vastaan, kutsutaan heikoksi tai kapeaksi tekoälyksi [14]. Nämä vaativat ihmisten osallistumista esimerkiksi algoritmien täsmentämisessä ja opettamisessa, jotta tekoäly antaisi tarkempia ja haluttuja vastauksia. Siinä missä heikko tekoäly suorittaa yhtä tehtävää sekä vaatii ihmisen osallistumista, vahva tekoäly on teoreettinen käsite ihmismielen tasoiselle tekoälylle [15]. Tällainen nykyään mahdoton, mutta tulevaisuuden tekoäly saattaa kyetä tiedostamaan itsensä, oppimaan sekä ratkaisemaan monenlaisia ongelmia. Vahvasta tekoälystä on muun muassa luotu hypoteettinen tulevaisuudenkuva nk. *teknologinen singulariteetti*<sup>2</sup>. Tässä tulevaisuudenkuvassa teknologinen kehitys on edennyt niin pitkälle, että tekoälyn kehittyminen ohi ihmisälyn aiheuttaa suuren kehitysaskelen tuoden hallitsemattomia ja peruuttamattomia vaikutteita elämäämme (kuva 1).

---

<sup>2</sup> Termi *singulariteetti* tulee astrofysiikasta ja sillä tarkoitetaan mustan aukon syntyessä sen keskelle muodostunutta tilaa, joka ei ole selitettävissä tai havaittavissa sen olemattoman pienen koon ja äärettömän tiheyden vuoksi [16].



**Kuva 1.** Singulariteetti-hypoteesia kuvaava graafi. Hypoteettinen tulevaisuudenkuva alkaa hetkestä, jolloin ihmisten älykkyys ja tekoäly risteävät. Leikkauspistettä kuvataan kuvassa termillä "the Singularity", eli Singulariteetti. [17]

Tekoälyn tutkijat jakavat tekoälyn viiteen osaan: oppimiseen, järkeilyyn, ongelman ratkaisuun, havaitsemiseen sekä kielen ymmärtämiseen [18]. Yksi tapa opettaa tekoälyä on yrittämisen ja erehtymisen kautta, jolloin vääränlaisen tuloksen saamisen takia tekoälyn algoritmin parametreja muutetaan tarkemman tuloksen saamiseksi. Järkeilyssä ja päättelyssä annetaan ratkaisu deduktiivisen tai induktiivisen<sup>3</sup> päättelyn ja tilanteen mukaan. Eri tilanteiden vuoksi tekoälyn pitäisi kyetä hyödyntämään tilanteeseen sopivaa päättelytapaa.

## 2.2 Tekoälyn osa-alueet

Tekoälyyn sisältyy osa-alueita kuten asiantuntijajärjestelmiä, koneoppimista, syväoppimista, luonnollisen kielen käsittelyä, neuroverkkoja ja sumeaa logiikkaa [14].

*Asiantuntijajärjestelmä* on järjestelmä, joka yrittää matkia ihmisen tai alalla toimivan organisaation arviointia sekä päätöksen tekoa. Järjestelmä kerää tietopohjaansa aiempia kokemuksia ja faktatietoja sääntöjen ja päätelmien luomiseksi. Järjestelmän tavoitteena ei ole korvata, vaan täydentää asiantuntemusta. [19]

<sup>3</sup> Deduktiivinen ja induktiivinen päättely ovat eri menetelmiä tehdä päätelmä. Deduktiivisessa päättelyssä todellisesta alkuoletuksesta tehdään päättely, kun taas induktiivisessa päättelyssä päättely tehdään yksittäisestä havainnosta.

*Koneoppiminen* on algoritmeja, joiden pyrkimyksenä on ymmärtää käyttäytyminen kerätyn materiaalin perusteella ilman sen luomista ohjelmoimalla. Koneoppiminen voidaan jakaa kolmeen alakategoriaan: ohjattuun (engl. "supervised learning"), ohjaamattomaan (engl. "unsupervised learning") ja vahvistusoppimiseen (engl. "reinforcement learning"). *Ohjatussa oppimisessa* läpikäytävästä datasta etsitään opetusdataan yhteensopivia tuloksia, jolloin tulos vastaa opetusdataa. *Ohjaamattomassa oppimisessa* algoritmi päättää itsenäisesti saadusta materiaalista tuloksen, ja sen avulla kielimallia voidaan opettaa suuremmalla raakadatalla ilman ihmisten osallistumista opettamiseen, mikä hidastaa opettamista [20]. *Vahvistusoppimisessa* algoritmille annetaan säännöt, joiden avulla sen täytyy löytää tuloksia, minkä jälkeen sille opetetaan, olivatko tulokset oikeanlaisia vai ei. [14,21]

*Syväoppiminen* on osa koneoppimista ja perustuu neuroverkkoon ("a neural network"). Kun perinteisessä koneoppimisessa käytettävää dataa täytyy käsitellä, jotta algoritmit tunnistavat etsittävän asian, neuroverkon avulla algoritmi voi käsitellä järjestämätöntä dataa kuten kuvia tai tekstiä automaattisesti; syväoppiva algoritmi kykenee määrittämään ja tunnistamaan haettavan asian yksityiskohdat itsenäisesti lähdemateriaalista. [22]

Kapea tekoäly keskittyy sille osoitetun tehtävän suorittamiseen. Miten generatiivinen tekoäly eroaa kapeasta tekoälystä on se, että se pystyy luomaan materiaalia perustuen dataan, millä sitä on opetettu [23]. Tällaista ehdotonta generatiivista tekoälyä kutsutaan kielimalliksi [24].

## 2.3 Neuroverkot

Keinotekoinen neuroverkko (engl. "artificial neural network") on verkkomainen, ihmisen aivoja jäljittelevä malli. Neuroverkkojen inspiraationa ovat olleet varhaiset aivojen aistinkäsittelyn mallit [25]. Verkko koostuu monista solmukohtista, neuroneista, jotka ovat yhdistyneet toisiinsa painotetuilla syöttölinkeillä [13]. Näistä muodostuu useita kerroksia, joissa kussakin on useampi neuroni, joihin edellisen tason neuronit yhdistyvät [5]. Tekoälyssä näitä kerroksia nimitetään termiä piilokerroksiksi. Kerrosten lisäksi kerrosten alkupäästä löytyy syötekerros ja loppupäässä ulostulokerros. Syötekerros on kerros, joka vastaanottaa verkkoon syötettävän tiedon kuten valokuvan. Piilokerrokset ovat neuroneita, jotka suorittavat laskutoimituksia. Ulostulokerros antaa piilokerroksissa luodut painotetut summat. [26]

Neuroni sisältää sisääntulo funktionin ("input function"), aktivointi funktionin ("activation function") ja ulostulon [13]. Yhdistettyjen neuronien välillä kulkevat väylät lähettävät signaaleja, ja jokaiselle niistä on määritetty kynnsarvot [27]. Kun signaali kulkee neuroverkon läpi, jokaisen neuronin kohdalla lasketaan lukuarvo. Tämä lukuarvon pitää olla suurempi kuin siitä neuronista lähtevän käytävän kynnsarvo, jotta kyseinen neuroni aktivoituu. Syöteaineistolle määritetään painoarvot, esimerkiksi yksittäisille sanoille omansa, joilla osoitetaan kyseisen sanan tärkeys ja vaikutus lopputulokseen. Näitä painoarvoja voidaan hienosäätää, mistä kerrotaan lisää seuraavassa luvussa.

Neuroverkkoja käytetään muun muassa konenäössä, lääketieteellisissä diagnooseissa, ja kuva- sekä puheentunnistuksessa [5]. Neuroverkon avulla konenäköä voidaan opettaa analysoimaan ja havaitsemaan esimerkiksi röntgenkuvien perusteella merkkejä keuhko-kuumeesta ja sydänsairauksista sekä syöpäsoluja. Googlen tutkimuksessa (Zhe Li et al.) [28] malli opetettiin 112 120 röntgenkuvalla, joissa oli kuvattu 14 erilaista todettua sairautta. Kehitetyn mallin avulla voidaan luoda kuvia, joissa lämpökarttamaisesti paikannettiin esimerkiksi rintakehän alueen sairaudet 74 % todennäköisyydellä. Rintasyöpään keskittyneessä tutkimuksessa (Yun Liu et al.) [29] 400 näytelevykuvasta havaittiin 92 % kasvaimista verrattuna patologian tekemään perusteelliseen tutkimiseen (73 %).

### 3. KIELIMALLIT

Kielimalli (engl. "language model", "LM") on algoritmi todennäköisyysjakaumien luomiseen sanasarjoille. Se ennustaa, millä todennäköisyydellä sana tai sanat esiintyvät sanajonossa. Sanasarjat koostuvat useammasta sanasta, joita kutsutaan "n-gram" sarjoiksi. Nimen *n*:llä viitataan sarjassa esiintyvien sanojen lukumäärään: kahdesta sanasta koostuva sanajonoa kutsutaan 2-gramiksi tai termillä "bigram", kolmen 3-gram ja niin edelleen [30]. Esimerkkinä 2-gram sarjasta on sanapari "punainen talo", ja vastaavasti 3-gram sarjasta "kiiltävän keltainen Kupla".

Ihmisen puhutun ja kirjoitetun tekstin kääntämisen käsittelyä tietokonejärjestelmille ymmärrettäväksi kutsutaan luonnolliseksi kielen käsittelyksi (engl. "Natural language processing", "NLP") [31]. Se hyödyntää laskennallista kielitiedettä, jolla tarkoitetaan ihmisen kirjoitetun ja puhutun kielen ymmärtämistä laskennallisesta näkökulmasta. Luonnollisen kielen käsittelyä voidaan hyödyntää monissa sovelluksissa, kuten puheentunnistuksessa, sanan merkityksen selittämisessä ja tunnistamisessa, semanttisessa analyysissä, sekä luonnollisen kielen luomisessa [32].

Jotta kielimallien tuloksista saadaan tarkempia, niitä täytyy opettaa. Tällaisia esiopetettuja ja säädettyjä kielimalleja kutsutaan esikoulutetuiksi kielimalleiksi (engl. "Pretrained Language Models", "PLM"). Esikoulutettu kielimalli on opetettu suurella määrällä dataa kuten tekstiaineistoa hyödyntäen [33]. Tämän jälkeen kielimallia optimoidaan hienosäätämällä, jotta se kykenee antamaan halutunlaisia vastauksia juuri siihen käyttöön, mihin esikoulutettua kielimallia halutaan käyttää. Yhtenä tarkoituksena on saada kielimalli oppimaan kielen rakenteet sekä mallit. Tämän avulla se pystyy vastaamaan loogisesti ja konkreettisesti käyttäjän syötteeseen. [34] Hienosäädön avulla kielimallia saadaan vastaamaan tietyn kategorian mukaisesti, kuten ammattitermejä käyttäen tai puhekielen mukaan.

Suuret kielimallit (engl. "large language models", "LLM") on nimitys kielimalleille, joita on esikoulutettu suurilla määrillä materiaalia. Se, missä menee kielimallien ja suurien kielimallien raja, ei ole yleisesti määritelty [35]. Kielimallien datan määrää kuvataan niiden lukumäärällä, joihin sisältyvät esimerkiksi yksittäiset kirjaimet, sanat, numerot sekä erikoismerkit. Näistä käytetään nimitystä tokeni (engl. "token").

Kielimallien kouluttamiseen käytetään sekä ohjattua ja ohjaamatonta oppimista, sekä muunnelmia näiden väliltä. Ohjatussa oppimisessa (engl. "supervised learning") hyödynnetään kahta datamassaa, joista ensimmäinen kokoelma sisältää merkittävää dataa ja toinen kokoelma testiä varten soveltuvaa, merkitsemätöntä dataa. Merkityllä datalla tarkoitetaan esimerkiksi eläinkuvien merkitsemistä niissä näkyvien eläimien perusteella, mitä kielimalli hyödyntää testidatan ymmärtämiseen. Merkitsemätöntä dataa tulkittaessa merkityllä datalla opetetulla kielimallilla saadaan tarkkuus, jota uudelleen ajamalla saadaan nostettua, kunnes virheiden määrä saadaan minimoitua mahdollisimman alhaiseksi. [36,37] Ohjaamattomassa oppimisessa (engl. "unsupervised learning") annettu lähdemateriaali on merkitsemätöntä, ja siinä syväoppivalle mallille ei anneta ohjeistusta materiaalin käsittelyyn [36]. Analysoimalla lähdemateriaalia neuroverkko pyrkii löytämään yhtenäisyyksiä ja struktuureja. Tämän avulla opetuksessa pystytään käyttämään suurempia määriä dataa, minkä vuoksi suurien kielimallien opettamisessa suositaan ohjaamatonta oppimista [38].

Tekoälytutkimukseen keskittyvä tutkimuskeskus OpenAI on hyödyntänyt kielimallien kouluttamisessa vahvistavaa oppimista ihmispalautteesta (engl. Reinforcement Learning from Human Feedback, RLHF). Tässä kouluttamistavassa ihmisen kirjoittama vastaus annetaan ensin mallivastauksena kielimallille ohjatulla opettamisella (engl. supervised learning). Tämän jälkeen kielimallin annetaan vastata samaan kysymykseen useasti, mistä ihmisarvioija arvioi vastaukset paremmuusjärjestyksessä. Tämän avulla luodaan palkitsemismalli. Tätä palkitsemismallia hyödyntämällä voidaan hienosäätää vastauksia toistuvasti ohjatun vahvistavan oppimisen avulla saaden malli käyttäytymään halutulla tavalla. Tätä kouluttamistapaa OpenAI on hyödyntänyt tutkimuskeskuksen luomassa GPT-3-kielimallissa. [39,40] Hyödyntämällä ihmisiä opettamisessa kielimallin vastauksista on saatu turvallisempia vähentämällä toksisuutta, hallusinaatioita<sup>4</sup> sekä muita haitallisia tuloksia, sekä onnistuttu lisäämään tuloksien osuvuutta.

### 3.1 Transformer-arkkitehtuuri

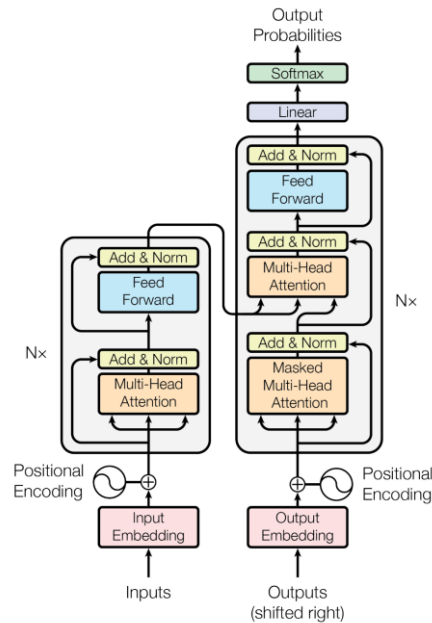
Jokainen kielimalli on rakennettu jonkin arkkitehtuurin mukaan. Yksi eniten käytetyistä suurien kielimallien arkkitehtuureista on Transformer-arkkitehtuuri. Transformer-arkkitehtuuri esiteltiin ensimmäistä kertaa Googlessa työskennelleiden tutkijoiden Ashish Vaswani et al. julkaisemassa tutkimuspaperissa "Attention is all you need" [42] vuonna 2017.

---

<sup>4</sup> Hallusinaatiolla tarkoitetaan kielimallien antamia vastauksia, joissa valheellisia tai keksittyjä väitteitä väitetään faktoiksi [39,41].

Transformer-arkkitehtuuri on todettu tehokkaaksi käännoistyössä, dokumenttien generoimisessa sekä syntaktisessa jäsentämisessä [43].

Transformer-arkkitehtuurista on muunnelmia, joiden kohdalla toistuu samankaltaiset komponentit: tokenien sisällyttäminen, tokenien sijainnin paikannuskoodaus, sisääntulotekstin kääntäjä (engl. "encoder"), tulosten sijainnin paikannuskoodaus ja niiden sisällyttäminen, koodinpurkain (engl. "decoder"), lineaarinen taso sekä softmax funktionin (kuva 2) [44].



**Kuva 2.** Transformer-arkkitehtuuria kuvaava diagrammi. Kuvakaappaus. [42]

Tokenien sisällyttäminen tapahtuu, kun tekoälylle syötetään dataa. Tokenit muutetaan koneille ymmärrettävään, numeraaliseen muotoon. Paikannuskoodauksen avulla mallille kerrotaan tokenien järjestys sekvenssi numerojoukkona, jotta kielimalli ymmärtää syöteen järjestyksen [45]. Tämän avulla sanoja ei tarvitse syöttää neuroverkkoon järjestyksessä [38] mitä ilman neuroverkot eivät ymmärrä tokenien järjestystä. Tästä muodostetaan matriisi, missä jokainen rivi esittää sekvenssin tokenia, mikä on lisätty sen sijaintitietoon (taulukko 1).

Taulukko 1. Esimerkki tokenien sijainnin paikannuskoodaamisessa muodostettavasta matriisista

Sarake		Tokenin järjestysluku		Tokenin sijainnin paikannuskoodauksen matriisi			
Minulla	→	0	→	$S_{00}$	$S_{01}$	...	$P_{0x}$
on	→	1	→	$S_{10}$	$S_{11}$	...	$P_{1x}$

punainen	→	2	→	$S_{20}$	$S_{21}$	...	$P_{2x}$
auto	→	3	→	$S_{30}$	$S_{31}$	...	$P_{3x}$

Sisääntulotekstin (engl. "input embedding") ja lähtöupotuksen (engl. "output embedding") avulla käyttäjän antama teksti käsitellään vektoriksi, jotta kielimalli ymmärtäisi sen sisällön ja saadakseen tarvittavan tiedon [46]. Transformer-arkkitehtuurissa encoder käyttää täysin näkyvää peittämistä (engl. "fully visible" attention mask), minkä avulla itsehuomioiva mekanismi saadaan kohdistamaan käsittelynsä mihin tahansa syötteeseen. Tämän avulla se voi tarkastella koko syötettä, jota hyödyntää tulosteessa. [47]

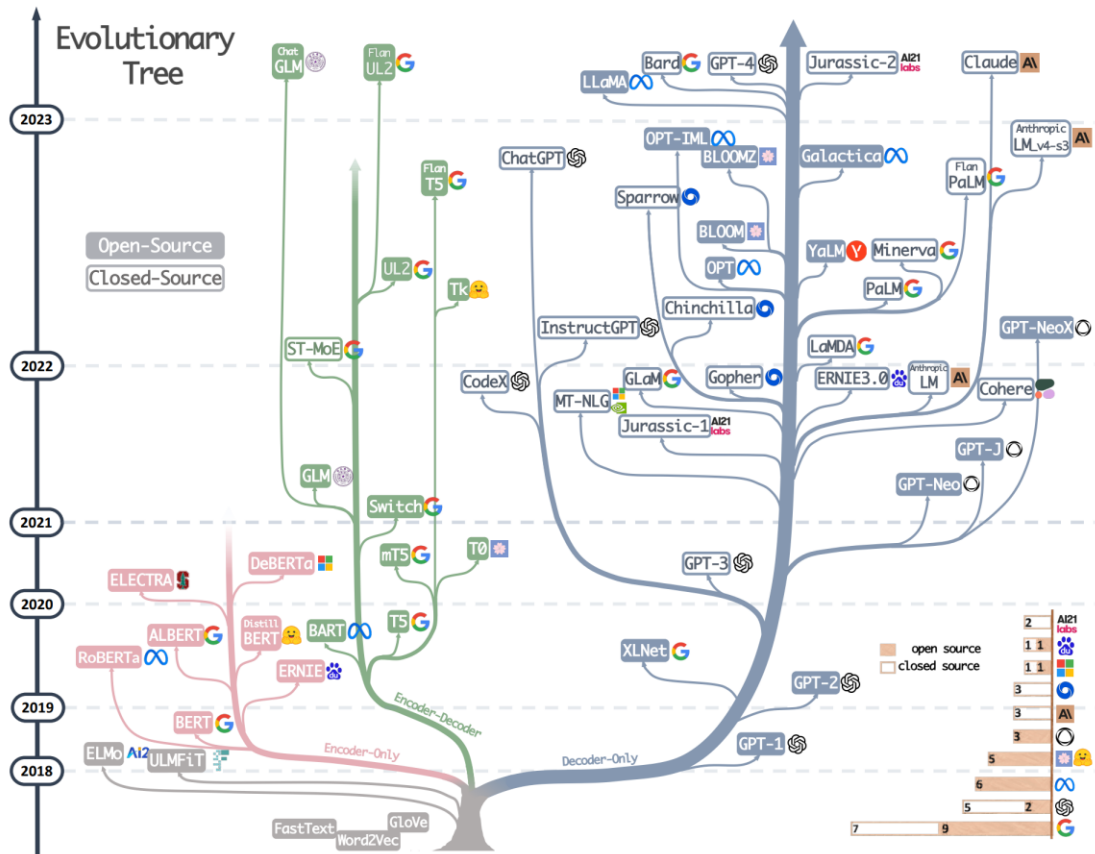
Decoder -osaa käytetään tuottamaan tulostus-sekvenssi, joka luodaan valitsemalla tokeni. Se syötetään takaisin kielimalliin, jotta saadaan tuotettua seuraava todennäköisin tokeni [47]. Tätä toistetaan, kunnes koko syöte on tulostettu. Lopulta hyödyntämällä Softmax<sup>5</sup> -funktionia, saadaan luotua lopulliset painoarvot [42]. Todennäköisyysjakauman diskreetillä muuttujalla voi olla useita mahdollisia arvoja, joista saadaan todennäköisyysjakauma Softmax-funktionilla [49].

Encoder-decoder-rakenteen lisäksi Transformerista on johdettu kaksi muuta rakennetta: "encoder-only" ja "decoder-only" (kuva 3). Encoder-rakenteesta ei huomioida decoder -osaa, jolloin siinä ainoastaan muutetaan käyttäjän syöte koneelle ymmärrettävään muotoon. Sitä käytetään etenkin silloin, kun halutaan luokitella käyttäjän syötettä esimerkiksi kategorioihin tai kun halutaan poimia kysymysvastaukset [50]. Eräs tällaisista kielimalleista on vuonna 2019 Googlen kehittämä BERT (Bidirectional Encoder Representations from Transformers) [51].

Kuten encoder -rakenne, decoder -rakenne hyödyntää ainoastaan Transformer -arkkitehtuurin koodinpurkain osaa. Tätä variaatiota ollaan käytetty yleisemmin kielimalleissa [52]. Esimerkkinä OpenAI:n kehittämä GPT, jonka Alec Radford et al. julkaisivat artikkelissa "Improving Language Understanding by Generative Pre-Training" [20].

---

<sup>5</sup> Softmax- eli normalisoitu eksponentiaalinen funktio on matemaattinen funktio, joka muuttaa vektorin sisältämät luvut arvoiksi 0 ja 1 väliltä. Tämän avulla luvut saadaan tulkittua todennäköisyyksinä. [48]



**Kuva 3.** Suurten modernien kielimallien kehittymistä kuvaava sukupuu. Pystyjana kuvaa tunnettujen kielimallien julkaisuvuoden. Kielimallien taustaelementit kertovat koodin avoimuudesta (taustaväri avointa lähdekoodia, valkoinen tausta reunaviivalla suljettua koodia). Värilliset (muut kuin harmaat) kuvaavat Transformer-arkkitehtuurin kielimalleja. Oikealla alareunassa pylväsdiaagrammi kuvaa eri yritysten ja instituutioiden julkaisemien kielimallien lukumäärää sekä niiden avoimuutta. [52]

### 3.2 GPT

GPT (Generative Pre-training Transformer) on tutkimus- ja käyttöönottoyhtiö OpenAI:n kehittämä kielimalliperhe. Kielimallina GPT hyödyntää Transformer-arkkitehtuuria ja sen decoder -muunnelmaa. Se on yleisopetettu ohjaamattomalla esikouluttamisella sekä parannetulla ohjatulla hienosäädöllä (engl. "supervised fine-tuning") [43]. GPT-kielimallit on opetettu ymmärtämään yleistä kieltä (engl. "natural language") ja koodia. Käyttämällä malleja, OpenAI:n tarjoamaa avointa rajapintaa tai heidän verkkopalveluaan ChatGPT:tä, GPT:tä voidaan hyödyntää muun muassa dokumenttien, koodin, käännöksiä ja tekstin luomiseen [53]. Kielimallia voidaan ohjata antamalla ohjeistuksia tekstisyötteellä, minkä avulla voidaan kertoa kielimallin käyttäytyvän tietyn tavoin vastatessa käyttäjille.

OpenAI on perustettu joulukuussa 2015. Blogikirjoituksessaan Greg Brockman et al. kertovat perustaneensa yrityksen pyrkimyksenä olla johtava instituutio, joka priorisoi hyvän lopputuloksen oman edun ajamisen sijaan tilanteessa, kun lähestytään ihmistasoista tekoälyä. Yrityksen tarkoituksena oli jakaa heidän tutkimuksiaan avoimesti muiden instituutioiden kesken. Blogissaan rahoittajina mainitaan muun muassa Reid Hoffman, Elon Musk, Jessica Livingston, Peter Thiel ja Amazon Web Services (AWS). Blogikirjoituksessa rahoittajien kerrotaan lahjoittaneen miljardi Yhdysvallan dollaria tukeakseen OpenAI:ta. [54,55]

OpenAI julkaisi ensimmäinen kielimallinsa version, GPT-1:n, vuonna 2018 julkaistun ”Improving Language Understanding by Generative Pre-Training” yhteydessä [43]. Kielimallin opetusdatana hyödynnettiin BooksCorpus- ja 1B Word Benchmark -tietokantoja. BooksCorpus-tietokanta on kerätty hyödyntämällä yli 10 000 ilmaista verkkokirjaa eri genreistä kuten fantasia sekä tiede [56]. Se sisältää 1,3 miljoonaa uniikkia sanaa. 1B Word Benchmark on Googlen julkaisema kokoelma kielimallintamista varten [57].

GPT-1:n jälkeen OpenAI on jatkokehittänyt heidän kielimalliaan. GPT-2 julkaistiin ensimmäistä kertaa vuonna 2019, minkä jälkeen siitä julkaistiin neljä eri kielimalli-versiota eri parametrimäärillä. Toukokuussa 2019 julkaistujen kielimallien parametrien määrät ovat 124 miljoonasta ylöspäin 1,5 miljardiin asti [58].

Vuonna 2020 julkaistu kolmannen sukupolven GPT-3 hyödyntää GPT-2-mallin arkkitehtuuria, mutta eri neuroverkkokuviolla (engl. ”sparse attention patterns”). Sen kuvio on tehty muuttumaan vaihtelevasti tiheästä harvoin [59]. Muuttuvan neuroverkon tason tiheyden avulla suuren datamäärän laskeminen on yhtä nopeaa tai nopeampaa verrattuna Transformer -mallin, kiinteätiheän tasoiseen verkkoon [60]. Tästä hyödytään etenkin silloin, kun kielimallin syöte on suuri, jolloin laskenta on raskasta. Julkaisussaan ”Language Models are Few-Shot Learners” Tom Brown et al. kertoo, että GPT-3:a varten kerättiin opetustietoa vuoteen 2019 saakka [59]. Tämän vuoksi kielimalli ei kykene kertomaan tai selittämään asioita, jotka ovat esiintyneet tai tapahtuneet vuoden 2019 jälkeen.

Kolmannen sukupolven GPT:stä on julkaistu kahdeksan eri muunnosta, joista GPT-3:na tunnettu malli ”GPT-3 175B” on koulutettu 175 miljardilla parametrilla [59]. Se on optimoitu tekstipohjaiseen keskusteluun luoden luonnollista kieltä [61]. Kielimallin rajoittavana tekijänä voidaan pitää kielimallin opettamiseen käytettyä dataa, jotka on julkaistu ennen lokakuuta 2019 eikä täten joissakin tapauksissa ole ajankohtaista.

GPT-3:n jälkeen on tullut sekä GPT-3.5- että GPT-4-versiot. Vuonna 2022 julkaistu GPT-3.5 [62] ja vuonna 2023 julkaistu GPT-4 [63] on koulutettu ohjatulla hienosäädöllä eikä

vahvistavalla oppimisella (engl. "reinforcement learning") kuten GPT-3. Yritys kutsuu GPT-4-versiota tehostetun kapasiteetin kielimalliksi, jolla on kehittyneempi päättelykyky ja joka kykenee antamaan monimutkaisempia ohjeita mahdollistaen luovemmat tulokset [64]. Sekä GPT-3.5-että GPT-4-versiot on opetettu materiaalilla, jotka on kerätty ennen syyskuu 2021 [61]. Vuoden 2021 dataan perustuvan GPT-4:n jälkeen OpenAI julkaisi heinäkuussa 2023 GPT-4:n kaikille saataville [65]. Tämän jälkeen tutkimuskeskus on julkaissut esittelymalleja, jotka voivat käsitellä 128 000 tokenia yhdessä syötteessä luodessaan vastausta. Alkuperäinen GPT-4-malli kykeni käsittelemään yhdessä syötteessä "vain" 8192 tokenia [66]. Näistä esittelymalleista kirjoitushetkellä uusim, GPT-4 Turbo (gpt-4-0125-preview), on opetettu joulukuuhun 2023 asti kerätyllä tiedolla.

Hyödyntämällä GPT-kielimalleja OpenAI on julkaissut eri tarkoituksiin sopivia sovelluksia. Tammikuussa 2022 OpenAI julkaisi InstructGPT-kielimallin, jonka pohjalla on hyödynnetty GPT-3-kielimallia [67]. Sen kouluttamisessa on käytetty vahvistavaa oppimista ihmispalautteesta (engl. "Reinforcement Learning from Human Feedback", RLHF). Saman vuoden marraskuussa OpenAI julkaisi kielimallia hyödyntävän keskustelusovelluksen ChatGPT:n.

### 3.3 PaLM

Google I/O 2022 -tapahtumassa Google esitteli uuden luonnollisen kielimallin PaLM (engl. "Pathways Language Model"). Se perustuu Transformers-arkkitehtuurin koodinpurkaimen, joka on koulutettu 540 miljardilla parametrilla ja kerrotaan kykenevän ajatusketjun kehoituksilla (engl. "chain-of-thought prompting") ratkaisemaan monivaiheisia päättelytehtäviä [68,69].

Syksyllä 2023 julkaistua tuoreempaa versiota PaLM 2:a kuvaillaan Googlen blogikirjoituksessa kehittyneemmäksi kielimalliksi monikielisyyden, päättelykyvyn ja ohjelmoinnin osa-alueilla [70]. Mallin kerrotaan olevan kykenevä ymmärtämään useampaa kuin sataa kieltä, osoittamaan kykyä loogiseen ja matemaattisten ongelmien päättelyyn sekä yleisesti tunnettujen sekä vähemmän tunnettujen ohjelmointikielien kuten Prolog:n tuottamiseen.

### 3.4 LaMDA

PaLM:in rinnalla Googlen tytäryhtiö Google DeepMind (myöh. DeepMind) kehittää muita kielimalleja, kuten LaMDA-kielimalliperhettä. LaMDA (Language Model for Dialogue Applications) on DeepMindin kehittämä kielimalliperhe, joka alun perin julkaistiin toukokuussa 2021 [71]. Helmikuussa 2022 julkaistussa artikkelissa Romal Thoppilan et al. [72] kuvaavat sen perustuvan Transformer-arkkitehtuurin decoder-osaan sekä olevan jatkokehitystä Googlen vuonna 2020 julkaistulle tutkimukselle ja Meena-keskustelumallille<sup>6</sup>. Opettamisessa on hyödynnetty julkista keskusteludataa ja muita julkisesti verkossa saatavilla olevia dokumentteja. Se on opetettu käyttäjän kanssa vuoropuheluun.

Toukokuussa 2022 järjestetyn Google I/O 2022 -tapahtuman yhteydessä Google julkaisi seuraavan sukupolven version LaMDA 2:n [68]. Myöhemmin helmikuussa 2023 Google julkaisi oman keskustelupohjaisen palvelunsa nimeltä Bard, joka hyödynsi optimisoitua versiota LaMDA-kielimallista [74,75]. Blogissa kerrotaan palvelun hyödyistä tuottavuuteen, luovuuteen ja uteliaisuuteen, vaikka kirjoittajat tunnustavat, ettei palvelu ole täydellinen sen satunnaisesti antamien virheellisten tietojen vuoksi (nk. hallusinointi).

Joulukuussa 2023 Google julkaisi kielimallin Gemini. Sen kerrotaan kykenevän hienos-tuneeseen päättelyyn, tekstin, kuvien ja audion ymmärtämiseen, sekä ymmärtämään että tuottamaan yleisempiä ohjelmointikieliä kuten Python ja Java. Blogissa Geminin väitetään kykenevän vastaamaan oikein 90 % kysymyksistä MMLU-testissä<sup>7</sup>, kun taas verrattavana ollut GPT-4 kykeni vastaamaan oikein 86,4 % kysymykseen. [76]

Gemini 1.0 -versiosta tarjottiin kolme eri optimisoitua mallikokoa: Nano, Pro ja Ultra. *Gemini Nano* on suunniteltu laitteilla käytettäväksi malliksi, *Pro* yleisesti käytettävänä mallina erilaisille tehtäville. *Ultra* on suunnattu monimutkaisten tehtävien ratkaisemiseen. Google julkaisi Gemini Pro -kielimallin Googlen tuotteille ja Gemini Nanon Googlen Pixel 8 Pro -älypuhelimille. Verkossa saatavilla oleva Gemini-keskustelupalvelu hyödyntää Gemini Pro -mallia. [76] Helmikuussa 2024 Google uudelleenbrändäsi Bard-palvelunsa nimellä Gemini, joka on saatavilla keskustelupohjaisena ohjelmana sekä selaimelle, että omana sovelluksena mobiililaitteille [77].

---

<sup>6</sup> Meena on Googlen kehittämä keskustelumalli, joka perustuu Transformer "seq2seq" -arkkitehtuuriin [73].

<sup>7</sup> MMLU (Massive Multitask Language Understanding) on 57 aihealueesta koostuva testi. Testi sisältää kysymyksiä matematiikasta, fysiikasta, historiasta, laista, lääkkeistä ja etiikasta. Testin tarkoituksena on testata sekä tietoa että ongelmien ratkaisukykyä. [76]

### 3.5 OpenAI Codex

OpenAI Codex on GitHub:n, OpenAI:n ja Microsoftin yhteistyössä kehittämä kielimalli [78], joka keskittyy käsittelemään koodia luonnollisesta kielestä. Se julkaistiin ensimmäistä kertaa kesäkuussa 2021 GitHub Copilot -sovelluksen kanssa. OpenAI Codex:n ensimmäinen versio perustuu OpenAI:n GPT-3-malliin, josta GitHub:n blogipalvelussa silloinen toimitusjohtaja Nat Friedman kirjoitti sen olevan ”huomattavasti tehokkaampi” kuin GPT-3. Perustelluiksi kerrotaan, että OpenAI Codex on opetettu tiedolla, joka pääasiassa keskittyy avoimeen lähdekoodiin. [79] GitHub on verkkopalvelu Git-versionhallintaa hyödyntäville projekteille, missä on tarjolla sekä avoimen lähdekoodin projekteja että lisensseillä suojattuja sovellusprojekteja.

Elokuussa 2021 OpenAI julkaisi parannetun version OpenAI Codex:sta [80]. OpenAI:n blogijulkaisussa (Zaremba et al.) kerrotaan, kuinka kielimalli ymmärtää tuhansia ohjelmointikieliä ja kykenee tulkitsemaan yksinkertaisia komentoja luonnollisella kielellä. Opetettuina kielinä OpenAI Codex:n kerrotaan tukevan monia kieliä, joista blogikirjoituksessa mainitaan muun muassa JavaScript, Go, Swift sekä Shell. Parannellun version kerrotaan olevan tarjolla suljettuna API-rajapintana, mutta luvattiin julkaistavan myöhemmin avoimena rajapintana.

## 4. TEKOÄLYT JA KIELIMALLIT PALVELUINA

Jotta kielimallien hyödyntäminen olisi käyttäjäystävällisempää sekä saatavilla suuremmalle määrälle potentiaalisia käyttäjiä, kielimallien hyödyntämiseksi on luotu erilaisia sovelluksia sekä työkaluja. Erilaisten kielimallia hyödyntävien työkalujen avulla voidaan luoda erilaista tietoa kuten kuvia, ääntä, videota, tai jopa 3D-malleja. Tässä diplomityössä keskitytään tekstipohjaisiin kielimalleihin sekä niitä hyödyntäviin sovelluksiin, jotka tuottavat tekstipohjaisia tulosteita. Pienkehitystyössä, johon palataan työn myöhemmässä osassa, ratkaistavat haasteet ovat pääasiassa tekstipohjaisia, minkä vuoksi tässä diplomityössä keskitytään tekstipohjaisiin kielimalleja hyödyntäviin työkaluihin ja palveluihin.

Kielimallien saatavuutta on parannettu julkaisemalla sovelluksia ja työkaluja, joiden avulla keskustelu kielimallien kanssa tapahtuu verkkopalvelun chat-keskustelun kautta. Tällaisia ovat esimerkiksi työhön rajatut OpenAI ChatGPT ja Google Bard. Tämän lisäksi saatavilla on työkaluja, joita voidaan liittää osaksi olemassa olevaa sovellusta lisäosana. Tällaisia ovat muun muassa ohjelmointikehityksessä käytettävät tekstieditorit ja niihin asennettavissa olevat lisäosat, joiden avulla kommunikointi kielimallien kanssa tapahtuu suoraan tekstieditorin näkymässä. Yksi tällaisista palveluista on GitHub Copilot, johon palataan ChatGPT:n ja Bard palveluiden lisäksi tässä luvussa.

Generatiiviset tekoälytyökalut, joilla luodaan tekstiä tai kuvia esimerkiksi tekstisyötteestä, ovat tulleet yleisempään tietoisuuteen ja käyttöön kielimalleja hyödyntävien ohjelmarajapintojen eli API:en yleistyessä<sup>8</sup>. Hyödyntämällä API:a poistetaan pakollisuus kielimallin lataamiselle sekä toiminnallisuuden pystyttämiseksi paikallisesti sen hyödyntämiseksi. Samalla kun sovellukseen lisätään toiminnallisuudet hyödyntävät kielimallia suoraan sovelluksesta käsin kuten tekstieditorissa, saadaan palvelu helpommin saataville poistamalla kielimallin avaamista erillisessä selainikkunassa. Samalla esimerkiksi tekstieditorien kohdalla tekstien ja tiedostojen analysointi kielimalleilla helpottuu.

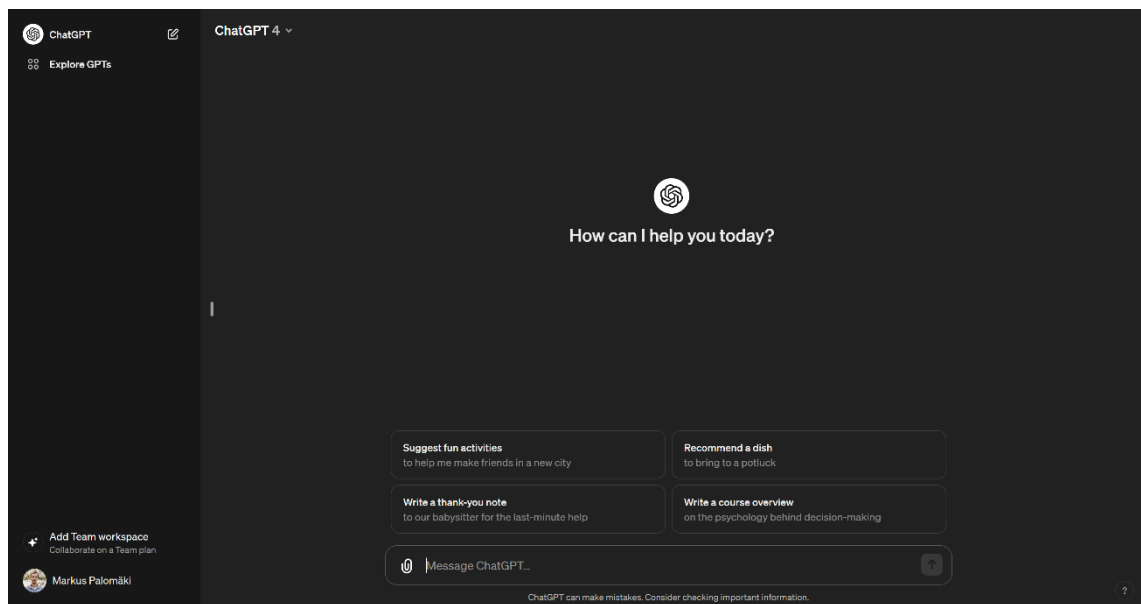
---

<sup>8</sup> Ohjelmarajapinta (engl. "Application Programming Interface", API) on keskusteluväylä pyyntöjen luomiseen ohjelmien tai komponenttien välille ilman pääsyä rajapinnan taustalla pyörivään ohjelman koodiin tai järjestelmään.

Alan nopeasta kehityksen tahdista kertoo sekä edellinen että tämä luku. Muun muassa työtä varten rajatuista palveluista moni sai uudistuksia. Kielimalleja opetetaan ja parannellaan jatkuvasti, mikä heijastuu myös palveluiden brändien muuttamisessa. Tässä yhtenä hyvänä esimerkkinä on Google Bard:n kehittyminen ja brändin muuttaminen vastaamaan Googlen kielimalliperhettä Geminiä.

## 4.1 ChatGPT

ChatGPT on OpenAI:n kehittämä keskustelusovellus, joka pohjautuu GPT-3.5:en ja sen uusimpiin versioihin [53,62]. Sen kerrotaan olevan InstructGPT:n sisarmalli ja hyödyntävän vahvistavaa oppimista ihmispalautteesta palkitsemiskielimallilla, jota hyödynnetään kielimallin hienosäädössä. OpenAI tarjoaa siitä eri palveluita, kuten keskustelupohjaisen verkkosovelluksen, joka tarjoaa käyttäjille mahdollisuuden kokeilla yrityksen kutsumaa tutkimusesikatselun (engl. research preview) mallia [81]. Kuukausimaksullinen versio ChatGPT Plus tarjoaa käyttäjälle mahdollisuuden käyttää GPT-4-versiota, joka muun muassa kykenee tulkitsemaan koodia suorittamalla sen [64]. Versio kykenee myös hyödyntämään käyttäjän lataamia tiedostoja.



**Kuva 4.** ChatGPT:n käyttöliittymä. Kuvakaappaus.

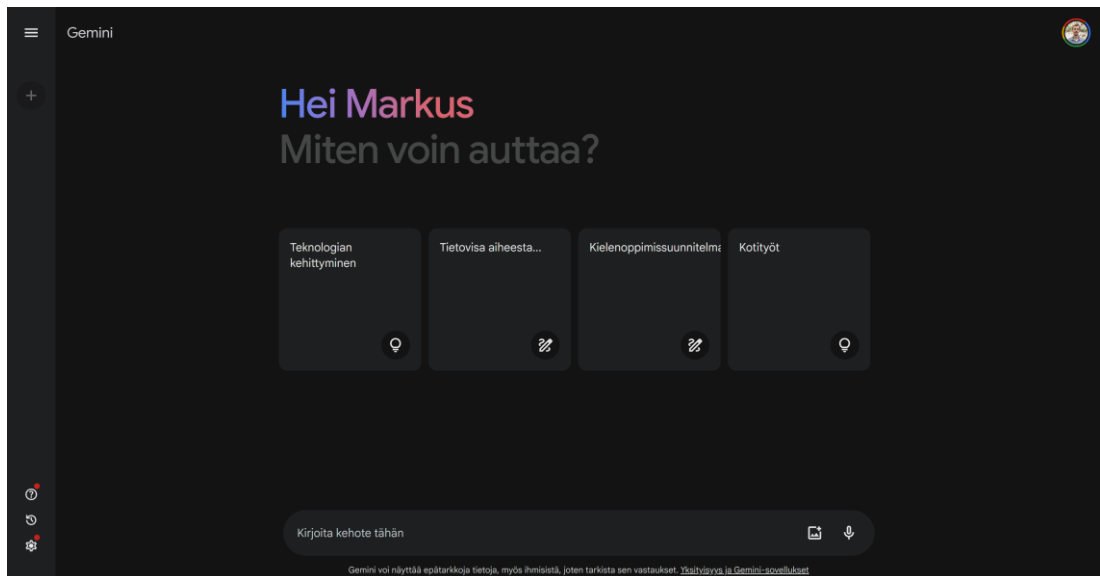
ChatGPT:n käyttöliittymä on yksinkertaistettu keskustelunäkymä, jossa keskustelunäkymä vie suurimman osan käyttöliittymän pinta-alasta (kuva 4). Vasemmalle sijoituvasta valikosta löytyvät perustoiminnot kuten asetukset ja tilausmahdollisuudet sekä aiemmin käydyt keskustelut listana ChatGPT:n luomilla otsikoilla. Ylhäällä sijaitsevasta pudotusvalikosta käyttäjä voi valita eri ChatGPT-version.

## 4.2 Google Bard

Google Bard on helmikuussa 2023 rajatulle testiryhmälle ja kuukautta myöhemmin beta-versiona julkisena julkaistu chat-pohjainen verkkosovellus kielimallin kanssa keskusteluun [74,75]. Se käytti Googlen kehittämää LaMDA-kielimallia ja myöhemmin Gemini-kielimallia. Helmikuussa 2024 Bard uudelleen nimettiin Geminiksi, minkä kautta haluttiin tuoda esille sen käyttämä kielimalliperhe Gemini ja tämän kehittyneemmät ominaisuudet [77]. Uudelleen nimeämisen yhteydessä palvelu poistui testausvaiheesta. Tässä työssä keskitytään sen hetkiseen versioon, Google Bardiin, joka oli saatavilla ja käytössä työn tekemisen aikana.

Google Bard on saatavilla verkko- ja mobiilisovelluksina. Sisäänkirjautumisen jälkeen käyttäjälle aukeaa näkymä, jossa käyttäjä voi syöttää kielimallille syötteen. Vasemmalla valikosta käyttäjä pystyy tarkastelemaan aikaisempia keskusteluja, lukemaan ohjeet, katsomaan tekemänsä toiminnot ja muokkaamaan asetuksiaan (kuva 5).

Verrattuna muihin työhön valittuihin työkaluihin (GitHub Copilot, OpenAI ChatGPT), Google Bard tarjoaa mahdollisuuden tarkistaa kielimallin antaman vastauksen painamalla vastauksen alaosassa olevaa Googlen logoa. ”Tarkista vastaus huolellisesti” painike hakee Googlen hakutuloksesta tietoja ja merkitsee tekstejä tarjoten niitä klikkaamalla lisätietoa ja linkkejä hakutuloksiin.



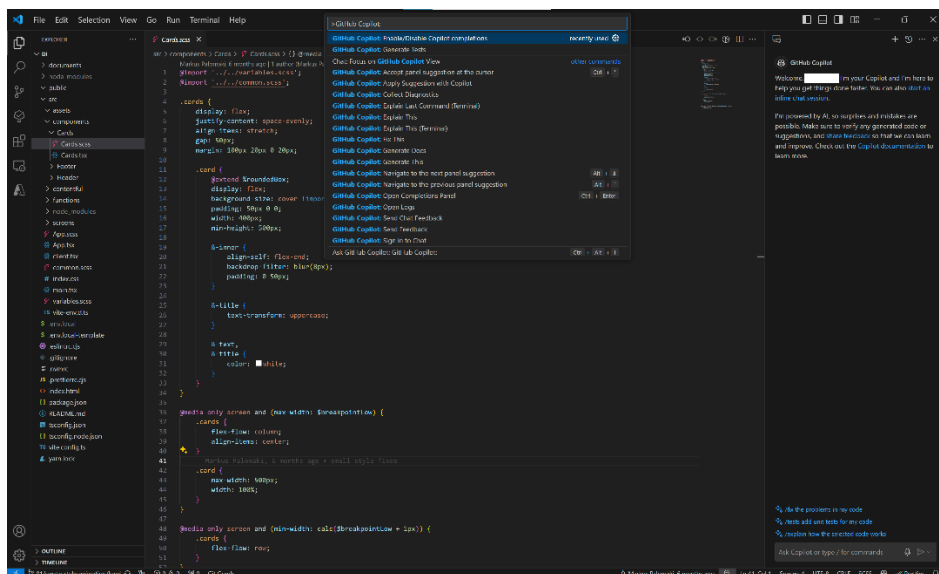
**Kuva 5.** Nykyisen Google Geminin (ennen Google Bard) käyttöliittymä. Gemini-päivitys ei tuonut ulkoasuun merkittäviä muutoksia. Kuvakaappaus.

### 4.3 GitHub Copilot

GitHub on vuonna 2008 perustettu [82] versionhallintapalvelu ohjelmointikoodin ja tiedostojen varastointiin pakettivarastoihin sekä julkisesti että yksityisesti. Se hyödyntää Git-versiohallintaa ja tarjoaa käyttäjille versiohallinnan lisäksi muun muassa CI/CD-automaatiot, tikettijärjestelmän sekä projektihallinnan [83].

Näiden toiminnallisuuksien rinnalle GitHub julkaisi kesäkuussa 2021 teknisen ensiversion palvelusta GitHub Copilot [79]. Palvelu on toteutettu lisäosaratkaisuna tekstieditorille ja tukee muun muassa suosittuja editoreita Visual Studio Code sekä the JetBrains suite of IDEs. Julkaistaessa GitHub Copilot käytti OpenAI Codex -kielimallia joka perustui GPT-3 -kielimalliin, mutta myöhemmin marraskuussa 2023 GitHub julkaisi blogissaan sen päivittyneen GPT-4 -versioon [84]. OpenAI Codexin kouluttamiseen GitHub on hyödyntänyt julkisia pakettivarastoja ja kertoo mallin tuntevan laajasti eri ohjelmointikieliä. Näistä yhdeksi suosituimmaksi ohjelmointikieliksi GitHub kertoo JavaScriptin. [78] Suosituimpia ohjelmointikieliä seuraava palvelu GitHub 2.0 kertoo JavaScriptin olevan neljänneksi suosituin ohjelmointikieli GitHub-palvelussa (9,6 %) [85]. Viiden suosituimman listalta löytyvät lisäksi Python (16,9 %), Java (11,7 %), Go (10,3 %) ja C++ (9,5 %).

Asentaessa GitHub Copilot -lisäosan Visual Studio Code -tekstieditorille käyttäjä saa käyttöönsä monia toiminnallisuuksia (kuva 6). Lisäosan avulla käyttäjä voi kysyä apua chat-näkymässä, saada koodiehdotuksia kirjoittaessaan koodia, parantaa tai pyytää selityksiä kursorilla valitusta tekstistä avatusta tiedostosta, saada ohjeita kaikkien projektitiedostojen perusteella ja korjata koodissa olevia ongelmia [86].



**Kuva 6.** Visual Studio Code käyttöliittymä. Oikeassa reunassa paneeli GitHub Copilot keskustelulle. Muokattu kuvakaappaus.

## 4.4 Hinnoittelu ja ominaisuudet

Kielimallien antamien vastauksien vertailun lisäksi on hyvä ottaa huomioon niiden tarjoamat ominaisuudet sekä hinnoittelut. Valituista kielimalleja hyödyntävistä sovelluksista on julkaistu eri versioita, joissa maksullisissa tai arvokkaimmissa tilaussuunnitelmissa on saatavilla enemmän ominaisuuksia. Tässä luvussa vertaillaan kolmen valitun palvelun, ChatGPT, Google Bard, ja GitHub Copilot, hintoja ja ominaisuuksia.

Työn varsinaisen tutkimuksen toteuttamisen jälkeen Google uudelleen julkaisi keskustelusovelluksensa nimellä Gemini. Tämän vuoksi tässä luvussa kirjoitetaan kirjoitushetkellä kerrotuista hinnoista ja ominaisuuksista Google Geminiä tarkastelemalla. Muutoksen yhteydessä Geministä tuotiin maksullinen versio saataville, josta kerrotaan tässä luvussa, mutta jota ei hyödynnetty varsinaisessa tutkimuksessa.

### 4.4.1 OpenAI ChatGPT

OpenAI tarjoaa ChatGPT-palvelusta neljä eri kuukausimaksullista tilaussuunnitelmaa (taulukko 2). Yksittäisille henkilöille on suunnattu ilmainen versio ChatGPT Free ja maksullinen (20 USD per kuukausi) maksava versio ChatGPT Plus. Ilmaisversio perustuu GPT-3.5-versioon, kun taas maksullinen versio antaa käyttäjälle käyttöön GPT-4-versio. GPT-4 versio kykenee hakemaan tietoa ”selaamalla” reaaliaikaisesti verkosta [87]. Ilmaisversion GPT-3.5 mallia on rajoitettu 8K-konteksti-ikkunaan<sup>9</sup>, kun taas Plus sallii pidempiä kehotteita 32k-konteksti-ikkunallaan (gpt-4-32k -kielimalli) [88]. Plus-tilaussuunnitelma antaa muita OpenAI:n kehittämiä työkaluja käyttöön kuten tekstistä kuvaksi -kielimallia (engl. ”text-to-image”) hyödyntävän työkalun DALL·E. Kyvykkäämpään dokumenttien lukemiseen ja niissä olevien tietojen käsittelyyn tarjolla on kielimalli Advanced Data Analysis. [89]

Ryhmille, tiimeille ja organisaatioille OpenAI tarjoaa Team- ja Enterprise-tilaussuunnitelmat. Nämä tarjoavat samat ominaisuudet kuten Plus-suunnitelma, mutta lisäävät työkaluja kuten käyttäjien hallintaan liittyen, SAML SSO -kertakirjautuminen, käyttöanalyysit ja OpenAI:n asiakaspalvelun. Enterprise-tilaussuunnitelman konteksti-ikkuna perustuu Team-suunnitelman 32K-konteksti-ikkunan sijaan 128K-konteksti-ikkunaan. Tämän perusteella Enterprise-suunnitelman ChatGPT hyödyntää GPT-4 Turbo -kielimallia, kun taas 32K-konteksti-ikkunan kielimalli perustuu gpt-4-32k-kielimalliin [88].

---

<sup>9</sup> Konteksti-ikkunalla (engl. ”context window”) tarkoitetaan tokenien lukumäärää, jonka kielimalli voi yhdellä kerralla ottaa analysoitavaksi. Esimerkiksi 32k-luvulla tarkoitetaan 32 000 tokenin syötettä.

Team-suunnitelmalle ilmoitetaan hinnaksi 25 USD per vuosi tai 30 USD per kuukausi per käyttäjä. Enterprise-suunnitelmasta kiinnostuneita ohjataan ottamaan yhteyttä OpenAI:n asiakaspalveluun kysyäksään lisää tilaussuunnitelmasta sekä sen hinnasta. Sen hinnasta OpenAI ei julkisesti jaa tietoa dokumenteissaan.

Taulukko 2. *OpenAI ChatGPT-tilaussuunnitelmat ja niiden ominaisuudet [89].*

	ChatGPT Free	ChatGPT Plus	ChatGPT Team	ChatGPT Enterprise
<b>Kenelle</b>	Yksityinen	Yksityinen	Yksityinen, ryhmät, tiimit, organisaatiot	Organisaatiot
<b>Hinta</b>	Ilmainen	20 USD/kk	25 USD/vuosi tai 30 USD/kk per käyttäjä	-
<b>Ominaisuudet</b>	GPT-3.5-kielimalli 8K-konteksti-ikkuna	GPT-4 32K-konteksti-ikkuna Reaaliaikainen tiedonhaku GPTs-ohjelmien luominen ja jakaminen Äänisyöte ja -lähtö OpenAI:n muut luomat työkalut -dokumenttien analysointi -kuvien generointi	Plus-tilauksen ominaisuudet Organisaation tason käyttöhallinta Työtilojen hallinta Tiimin kesken GPTs-ohjelmien jako SOC 2 Type 1 -noudattaminen	Team-suunnitelman ominaisuudet GPT-4 Turbo 128K-konteksti-ikkuna SAML SSO Analytiikka OpenAI:n tarjoama asiakaspalvelu Yrityksen maininta ChatGPT-kotisivuilla

Plus-, Team- ja Enterprise-tilaussuunnitelman käyttäjille OpenAI tarjoaa mahdollisuutta luoda kustomoituja GPT-malleja tiettyihin käyttötarkoituksiin. Näille räätälöidyille sovelluksille OpenAI on antanut nimen "GPTs". GPTs-ohjelmien avulla käyttäjä voi kustomoida ChatGPT-keskustelun antamalla sille aloituskehotteen, jonka mukaan kielimalli vastaa käyttäjälle. OpenAI:n blogissa [90] kerrotaan, kuinka keskustelupohjainen kielimalli voidaan räätälöidä neuvomaan lautapeliä pelaamisessa, auttamaan lapsia matematiikan laskutehtävissä tai tarrojen luomisessa. GPTs-ohjelmia voidaan jakaa julkisesti muille käyttäjille tai vain tiimin ja organisaation kesken Team- ja Enterprise-tilaussuunnitelman tarjoaman ominaisuuden avulla.

#### 4.4.2 Google Bard

Google Bard on käyttäjälle ilmainen palvelu, jonka käyttäminen vaatii sisäänkirjautumisen henkilökohtaisella Google-tunnuksella. Lisävaatimuksena Googlen tarjoamissa dokumenteissaan kerrotaan, ettei tunnus saa olla yhdistetty Googlen tarjoamaan Family Link -palveluun [91]. Ilmainen Bard tarjoaa perustoimintoinaan chat-pohjaisen keskustelun ja kuvien analysoinnin, jotka perustuvat Gemini Pro 1.0 -kielimalliin [77]. Jotta Bardia voidaan käyttää koulun tai työpaikan sähköpostiosoitteella, tilien ylläpitäjien täytyy sallia sovelluksen käyttö.

Gemini-kielimalliperheestä on saatavilla Gemini Advanced -versio, joka käyttää Googlen tehokkaampaa 1.0 Ultra -kielimallia (taulukko 3) [92]. Se on saatavilla yhdessä Google One AI Premium -sopimuksen kanssa (22,99 euroa per kuukausi). Tilaussopimus tuo muita ominaisuuksia käyttäjän Google-tilille, kuten muun muassa Geminin Googlen sähköpostiin (Gmail) ja Google Docs -tekstinkäsittelyohjelmiin. Lisäksi palvelu antaa käyttäjälle kaksi teratavua pilvitallennustilaa pilvipalveluun Google Drive ja muita Google One Premium -tilauksen tarjoamia ominaisuuksia [93].

Taulukko 3. *Google Gemini ja maksullinen Gemini Advanced*

	<b>Gemini</b>	<b>Gemini Advanced</b>
<b>Kenelle</b>	Yksityinen	Yksityinen, oppilaitos, organisaatio
<b>Hinta</b>	Ilmainen	Osa Google One AI Premium-tilaus-sopimusta (22,99 €/kk)
<b>Ominaisuudet</b>	Keskustelu	Gemini Ultra 1.0
	Kuvan analysointi	Gemini Googlen sovelluksissa
	Gemini Pro 1.0 -kielimalli	Kaksi teratavua tallennustilaa

#### 4.4.3 GitHub Copilot

GitHub tarjoaa ChatGPT:n kaltaisesti kolme kuukausimaksullista tilaussuunnitelmaa: Copilot Individual, Copilot Business ja Copilot Enterprise [94] (taulukko 4). Individual -suunnitelma on tarkoitettu yksittäisille kehittäjille, freelancer, opiskelijoille ja pedagogeille. Sillä on 30-päivän kokeilujakso [95], jonka jälkeen tilaus muuttuu kuukausitilaukseksi (10 USD per kuukausi). Kuukausittain maksettavan tilauksen rinnalle GitHub on tuonut 100 USD maksavan vuositilauksen.

Varmistetuille opiskelijoille, pedagogeille sekä avoimen lähdekoodin kehittäjille Copilot Individual on ilmainen [94]. Alemman tason tilaussuunnitelma tarjoaa keskustelupohjaisen kielimallin, joka on kontekstittietoinen. Se muun muassa tukee ohjelmoinnissa tuntien ja opastaa virheisiin ja tietoturvasuuteen liittyvissä asioissa, antaa reaaliaikaisesti palautetta ja lisää kommenttirivejä koodiin kysyttäessä. GitHub-käyttäjäprofiilin asetuksista

käyttäjällä on myös mahdollisuus sallia tai estää Copilotin käyttämästä julkisia tietovaroja.

Taulukko 4. *GitHub Copilot tilaussuunnitelmat [78]*

	<b>Copilot Individual</b>	<b>Copilot Business</b>	<b>Copilot Enterprise</b>
<b>Kenelle</b>	Yksittäinen kehittäjä, freelancer, opiskelija, edagogi	Organisaatiot	Organisaatiot
<b>Hinta</b>	Ilmainen* 10 USD/kk 100 USD/v <i>*Ilmainen varmistetuille opiskelijoille pedagogeille ja avoimen lähdekoodin ylläpitäjille</i>	19 USD/kk per käyttäjä	39 USD/kk per käyttäjä
<b>Ominaisuudet</b>	Chat-keskustelu Copilot komentokehoteissa Koodipätkäkokoelma Kyky estää julkisista koodista luodut koodiehdotukset Koodieditoriin yhdistettävyys	Individual ominaisuudet Organisaation tason käyttöhallinta Yksittäisten tiedostojen poissulkeminen Käyttöhistoria	Individual ominaisuudet Organisaation tason käyttöhallinta Yksittäisten tiedostojen poissulkeminen Käyttöhistoria Copilot Chat Vetopyyntöjen tiivistelmän luominen

Kalliimmat tilaussuunnitelmat Business ja Enterprise ovat suunnattu organisaatioille, jotka haluavat hyödyntää palvelua. Nämä tilaussuunnitelmat tarjoavat Individual-tilaukseen verrattuna lisää ominaisuuksia organisaation, hallinnon ja käytäntöjen tasolla. Näitä ominaisuuksia ovat muun muassa organisaation tason käyttöhallinta. Tilaussuunnitelman käyttäjällä on mahdollisuus sulkea yksittäisiä tiedostoja pois, jotta Copilot ei pysty käsittelemään niitä. Muita ominaisuuksina Copilot tarjoaa käyttöhistorian, Copilot Chat -keskustelunäkymän GitHub-verkkosivuilla ja vetopyynnöstä (engl. "pull request") tiivistelmän luomisen [78].

## 4.5 Tietoturva, data ja oikeudet

Kielimallit ja niitä tarjoavat työkalut herättävät kiinnostusta niin yksittäisten henkilöiden kuin organisaatioiden tasolla. Samalla niiden käyttö voi herättää tietoturvakysymyksiä. Etenkin jos palvelu tarjotaan ilmaiseksi kokeiltavaksi tai käytettäväksi, millaista tietoa käyttäjästä kerätään ja mihin sitä käytetään? Millaiset oikeudet käyttäjällä on syötteen lisättyyn dataan, ja omistaako palvelun tarjoaja kielimallin kirjoittaman vastauksen? Organisaatioiden ja yritysten tasolla huolta saattaa herättää, voiko vain sisäiseen käyttöön osoitetut dokumentit tai ohjelmistokoodit olla vaarassa joutua osaksi kielimallien opetusdataa ja sitä tai muuta kautta ulkopuolisten käsiin.

OpenAI kertoo **ChatGPT**-sovelluksesta käyttöehdoissaan yleisesti [96], että käyttäjä omistaa syötteet sekä kielimallien antamat tulokset. Käyttöehdoissa jatketaan muistuttamalla, ettei syöte välttämättä ole aina uniikki, koska toiset käyttäjät voivat saavat vastaavanlaisen tai saman kielimallin luoman tuloksen. Tämä kerrotaan johtuvan tekoälyn luonteesta ja OpenAI:n palveluista. Lisäksi käyttöehdoissa mainitaan, että sisältöä voidaan hyödyntää OpenAI:n palvelujen tuottamiseen, ylläpitoon, kehittämiseen ja parantamiseen. Käyttäjillä on mahdollisuus kieltää datan käyttö esimerkiksi kielimallien opettamiseen, jonka voi ilmoittaa heidän tarjoaman yksityisyysportaalin kautta [97].

Lisäksi tukisivuilla [98,99] kerrotaan, ettei Team- ja Enterprise-tilaus suunnitelman käyttäjien työtilojen (engl. "workspace") tietoja käytetä opettamiseen. Enterprise-tilaus suunnitelman sivulla kerrotaan myös, ettei liiketoimintaan tai keskusteluja käytetä opetuksessa. Enterprise-tilien ylläpitäjä pystyy kontrolloimaan, kuinka kauan käyttäjien tietoja säilytetään palvelussa [100].

Turvallisuusportaalissa [101] OpenAI listaa yksityisyysuojina CCPA, GDPR, SOC 2 - ja SOC 3. CCPA (California Consumer Privacy Act) ja GDPR (General Data Protection Regulation) ovat yksityisyysuojan lakeja Kalifornian ja EU-alueella. Näiden tarkoituksena on lisätä läpinäkyvyyttä siitä, kuinka yritykset keräävät ja käyttävät tietoa, ja kuinka käyttäjillä on mahdollisuus vaikuttaa tiedon keräämiseen ja säilyttämiseen [102]. SOC 2 ja 3 -tarkastusraportit raportoivat palveluorganisaation turvallisuudesta, saatavuudesta, luottamuksellisuudesta ja yksityisyyteen liittyvistä valvontatoimista [103]. Datavälittäminen käyttäjien ja palvelimien välillä kerrotaan olevan salattuja. Näiden myönnettyjen sertifikaattien lisäksi OpenAI mainitsee, että data on salattu AES-256 -salauksella [100].

**Googlen Gemini** -palveluun liittyvällä tukisivulla [104] avataan lyhyesti sitä, kuinka käyttäjän dataa käsitellään Geminissä. Googlen kerrotaan käyttävän käyttöhistoriaa, paikkatietoa ja palautetta tarjotakseen, parantaakseen ja kehittääkseen Googlen tuotteita ja palveluita sekä koneoppimista. Sivulla pyydetään käyttäjiä olemaan jakamatta syötteisiin

luottamuksellisia tietoja tai muitakaan arkaluonteisia tietoja, joita käyttäjä ei haluaisi jaettavan tai näytettävän muille. Syynä tälle Google kertoo ihmisarvostelijat, jotka voivat lukea, merkitä ja käsitellä Gemini -palvelussa kirjoitettuja syötteitä. Tämä tehdään osana tuote- ja koneoppimisen kehitystä.

Keskustelujen poistaminen onnistuu käyttäjien toimesta. Jos syöte on käsitelty ihmisarvostelijan toimesta, minkä jälkeen käyttäjä poistaa syötteen, se säilyy kolmen vuoden ajan palvelussa. Tällöin säilytetty keskustelu anonymisoidaan poistamalla tieto käyttäjästä. Syötteitä ja lähetettyjä medioita käyttäjä voi hallita Googlen tarjoamassa palvelussa Gemini Apps Activity.

Googlen tarjoamista dokumentaatioista selvitettiin tietoa syötteiden ja kielimalleilla generoitujen vastauksien omistajuudesta, mutta tätä tietoa ei löytynyt. Tietoa haettiin sekä silmäilemällä että hyödyntämällä selaimen tekstin hakutoimintoa käyttäen aiheeseen liittyviä avainsanoja.

**GitHub Copilot** -palvelun tietoturvan ja tiedon keräämisestä GitHub kertoo dokumentissa [78], että Individual-tilaussuunnitelman käyttäjistä kerätään tietoa tuotteen tarjoamiseen, analysoimiseen ja tuotekehitykseen. Tiedot sisältävät tietoja muokkaustoimenpiteistä kuten kirjanpidon kielimallin ehdottamien ehdotuksien hyväksymisestä tai estämisestä sekä virhetietojen keräämisestä. Nämä voivat sisältää henkilökohtaista tietoa pseudonymisoidusti salattuna<sup>10</sup>.

Toimiakseen tekstieditorissa Copilotilla käyttäjän luoma kehote sisältää tekstitiedoston sisältöä kuten ohjelmointikoodia, joka lähetetään GitHubille. Ilman tätä kielimalli ei voi luoda ehdotuksia käyttäjälle. Kerättyä tietoa käytetään kielimallin kehittämiseen sekä sen algoritmien hienosäätöön ja optimoimiseen. Dokumenteissaan GitHub kertoo, ettei yksityistä koodia jaeta ehdotuksena muille käyttäjille.

Annetun vastauksen sisällöstä GitHub kirjoittaa luottamuskeskuksessa [106], että GitHub ei vaadi itselleen annettujen ehdotuksien (kielimallien luomia vastauksia) omistajuutta. Lisäksi sivulla kerrotaan, että jos ehdotus on jotain sellaista, mitä voidaan omistaa, GitHub ei tule vaatimaan sen omistajuutta. GitHub painottaa, että useammalla käyttäjällä on mahdollisuus luoda vastaavanlaisia tuloksia keskenään. Käyttäjällä on lopullinen vastuu siitä, että generoitu vastaus on tekijänoikeuksiltaan sellainen, että sitä voidaan käyttää.

---

<sup>10</sup> Pseudonymisoitua tietoa käyttäjästä ei pystytä enää yhdistämään tiettyyn henkilöön suoraan ilman lisätietoja. Anonymisoitu tieto taas on irrotettu käyttäjän tiedoista täysin tehden mahdolltomaksi yhdistää henkilöä anonymisoituun tietoon. [105]

## 5. JATKUVA SOVELLUSKEHITTÄMINEN JA PIENKEHITYS

Työssä tutkitaan kielimallien hyödyntämistä sovelluskehittäjän työkaluna. Jatkuvassa sovelluskehittämisessä projektin pienimuotoista kehittämistä voidaan kutsua pienkehittämiseksi. Pienkehittäminen voi olla osa asiakkaille myytyä ylläpitopakettia, jossa jo kehitetyn palvelun ylläpitoon sekä jatkokehittämiseen on sovittu rajallinen tuntimäärä kuukautta kohden. Tämä voidaan määrittää tunti- tai kuukausiperusteisesti, esimerkiksi viiden työpäivän mukaan (7,5 h per päivä, yhteensä 37,5 tuntia). Tämä mahdollistaa sen, että pienkehityksessä toimivalla sovelluskehittäjällä voi olla useita pieniä projekteja käynnissä samanaikaisesti.

Jatkuva sovelluskehittäminen ja pienkehittäminen voidaan käsittää eri tavoin eri organisaatioissa. Palveluina voidaan tarjota esimerkiksi asiakaspalvelu ongelmaraporttien kuten tikettien vastaanottamiseen, niiden käsittelyyn ja ratkaisemiseen. Käytettyjen teknologioiden mukaan sovelluskehittäjällä voi olla vastuullaan laajasti erilaisia teknologioita tai pilvipalveluita, rajapintoja ja tietokantoja. Web-sovelluskehittäjälle voi tulla tilanne, jossa hänen täytyy ymmärtää datatieteeseen liittyviä asioita. Monien projektien ja teknologioiden vuoksi sovelluskehittäjien tarvitsee tietää ja olla valmiina omaamaan uutta tietoa nopeasti.

Monessa pienkehitys projektissa mukana oleva sovelluskehittäjä saattaa kokea ylimääräistä kuormaa. Työkuormaa voivat kasvattaa muistettavien tietojen määrä, samanaikaisten työn alla olevien ja odottavien tehtävien lukumäärä, työskentelyn katkonaisuus useamman projektin vuoksi tai jatkuvan tietotulvan ja uuden oppimisen tarve.

Etenkin pienkehityksessä toimiva sovelluskehittäjä voisi hyötyä kielimallien hyödyntämisestä osana projektityöskentelyä. Kielimallit ja niitä hyödyntävät sovellukset ja palvelut voivat tukea sovelluskehittäjää tiedon etsimisessä, analysoimisessa, virheiden tunnistamisessa ja ominaisuuksien toteuttamisessa. Sida Peng et al. tutkimuksessa ”*The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*” [107] kirjoittajat kertovat kehittäjille suunnatusta kyselystä, jonka mukaan GitHub Copilotin käyttäminen lisäsi keskimäärin 35 % tuottavuutta ja lyhensi tehtävien valmiiksi saamisen aikaa 55,8 %.

## 6. KIELIMALLIEN HYÖDYNTÄMINEN OSANA PIENKEHITYSTÄ

Diplomityössä tutustuttiin eri kielimalleja tarjoaviin organisaatioihin ja heidän tarjoamiin palveluihin, sekä niiden hyödyntämiseen tehtävien ja ongelmien ratkaisussa. Tutkimuksessa tutkittiin tekoälyjä ja kielimallien hyödyntämistä osana pienkehitystä. Työkalujen valinta rajattiin työkaluihin ja palveluihin, jotka pohjautuvat verkkopalveluihin ja ovat täten helposti jokaisen saatavilla. Tämän avulla palveluiden käyttö ei vaadi käyttäjältä tehokkaita laitteistoja tai kookasta tallennustilaa, joita laitteella paikallisesti suoritettava kielimalli saattaa vaatia.

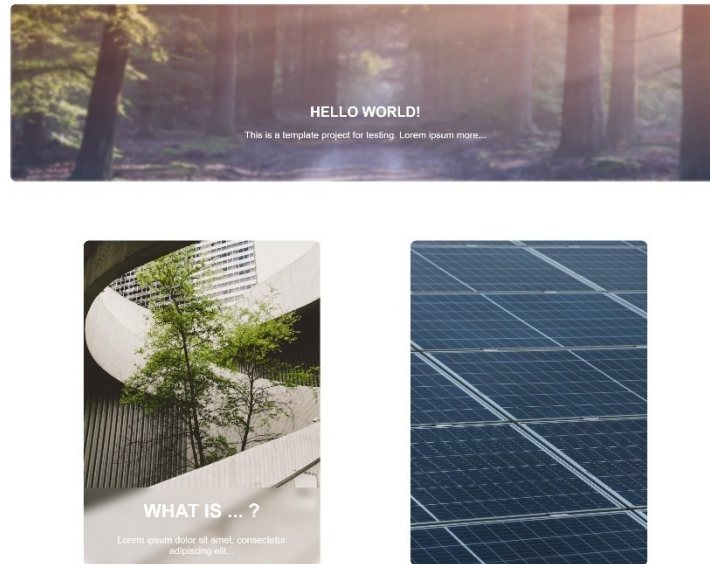
Työssä käytetyt kielimallit perustuvat vuonna 2017 Googlen kehittämään Transformer-arkkitehtuuriin. Näistä ChatGPT ja GitHub Copilot käyttävät OpenAI:n luomaa GPT-kielimalleja ja siitä luotuja eri versioita, kun taas Googlen Bard perustuu heidän luomaan LaMDA-kielimalliperheeseen.

### 6.1 Ratkaistavat tehtävät

Kielimalleilla ratkaistavia haasteita pyrittiin luomaan laajalta näkökulmalta katsottuna. Haasteiden vaikeustasot ja laajuudet olivat erilaiset, mutta sellaisia mitä sovellus- ja pienkehityksessä voidaan tavata. Haasteet keskittyivät ohjelmointiin.

Jokainen testi suoritettiin kullekin kielimallille tai sovellukselle mahdollisimman samalla tavalla. Sovelluksille syötetty aloitussyöte pidettiin samana, millä haluttiin pitää lähtökohta tasavertaisina. Ratkaistavat haasteet pyrittiin toteuttamaan kolmen syötteen avulla, ja jokaiset syötteet riippuivat edellisestä saadusta vastauksesta. Syötteiden kielinä käytettiin englantia. Jokaisesta syötteestä sekä vastauksesta, ja niistä nousseista havainnoista, kirjattiin ylös huomiot. Viimeisen kolmannen syötteeseen saadun vastauksen jälkeen testattiin ratkaisun toimivuutta ja sitä, kuinka paljon muutoksia jouduttiin tekemään halutun lopputuloksen saamiseksi.

Työtä varten luotiin paikallisesti pystytetty, Vite-kehitysympäristössä<sup>11</sup> toimiva React.js-kehystä<sup>12</sup> käyttävä verkkosovellus (kuva 7). Sovellus käytti Contentful-sisällönhallintajärjestelmää<sup>13</sup> (engl. ”Content Management System”) sisällön lisäämiseen ja päivittämiseen. Verkkosovelluksen rakenne koostui yhdestä näkymästä ja kolmesta komponentista: Header, Cards ja Footer. Näille kullekin luotiin oma tyylitiedosto.



**Kuva 7.** Ruudunkaappaus toteutetusta verkkosovelluksesta. Testiä varten luotu yksinkertainen käyttöliittymä hyödynsi React.js JavaScript-kehystä sekä Contentful-sisällönhallintajärjestelmää. Kuvakaappaus.

Ratkaistavia haasteita luotiin kaksi kappaletta tyyliaanimaation luomisesta sekä testiympäristön pystyttämiseen ja testien luomiseen. Haasteiden aikana dokumentoitiin sen hetkiset kielimallien versiot, niistä saatavilla olleet lisätiedot, havainnot haasteiden ratkaisemisen ajalta ja syötteiden avulla luotu lopputulos. Jos kolmannen vastauksen jälkeen lopputulos ei ollut odotetun kaltainen, tutkittiin kuinka pienellä tai suurella muutoksella saatiin toteutettua haluttu lopputulos toiminnallisuudelle.

Testiä varten luotu projekti sekä haasteet varastoitiin avoimen lähdekoodin Git-versiohallintajärjestelmällä. Jokaiselle testattavalle kielimallia hyödyntävälle sovellukselle luotiin Git-haara, ja jokaiselle vaiheelle dokumentteineen luotiin oma commit-sitouma. Tämän avulla toteutettuja haasteita pystyttiin tutkimaan ja vertailemaan jälkikäteen sekä dokumentoimalla että koodin tasolla.

<sup>11</sup> Vite on avoin frontend-kehitystyökalu. Saatavilla: <https://github.com/vitejs/vite>

<sup>12</sup> React.js on komponenttipohjainen JavaScript-kehys käyttöliittymien toteuttamiseen. Saatavilla: <https://react.dev/>

<sup>13</sup> Saatavilla: <https://www.contentful.com/>

## 6.2 Haaste 1: Tyylianimaation luominen

Haasteen tehtävänä oli luoda animaatio verkkosovellukseen, joka aktivoitui hiirtä liikuttamalla. Tällä haluttiin suurentaa kuvaa, mikä oli asetettu Card-komponenttiin. Animaatiosta haluttiin nopea ja sen kestoksi haluttiin 0,5 sekuntia. Lisäksi vaatimuksena oli kuvan koon muuttuminen animaation aikana 10 % suuremmaksi. Lopullisen tuloksen haluttiin vaikuttavan ainoastaan Card-komponentin sisälle asetettuun kuvaan eikä Card-elementin kokoon. Kuvan haluttiin pitävän kuvasuhteen oikeana animaation aikana.

Malleille syötettiin alkusyötteenä sama kehoite sisältäen lyhyesti käytössä olevista tekniikoista ja tavoitteesta sekä hetkiset elementit ja elementtien tyylimääritelmät (liite A). Lopputulosta arvioitiin toiminnallisuuden ja toteutuksen perusteella. Tämän jälkeen jäljelle jääneet ongelmakohdat pyrittiin ratkomaan ilman kielimallin apua dokumentoiden listaten, kuinka paljon työtä halutun lopputuloksen saavuttaminen lopulta vaati kolmen vastauksen jälkeen.

## 6.3 Haaste 2: Testiympäristön ja testien luominen

Ratkaistavana haasteena oli toteuttaa React-kehystä käyttävälle verkkosovellukselle yksikkötestit. Aloituvaiheessa verkkosivuprojektille ei ollut luotuna ympäristöä testien suorittamiselle eikä vaadittuja NPM-paketteja<sup>14</sup> asennettuna. Kuten ensimmäisessä haasteessa, ensimmäinen kehoite pidettiin kaikille sovelluksille samana. Siinä kuvataan lyhyesti mikä on tehtävä, mitä tekniikoita on jo sillä hetkellä käytössä ja kysymys koskien mitä tarvitaan testien luomiseen:

```
I need to create tests for the React project. The project uses Vite as build tool and contentful-cli to fetch data from Contentful. Yarn works as a package manager. What I need to create tests?
```

Syötteeseen kirjoitettiin tarkoitus luoda testejä React-projektille, joka hyödyntää Vite-kehitystyökalua. Tiedon hakemiseen projekti käyttää contentful-cli -pakettia, ja paketinhallintana käytetään NPM-pakettirekisteriä hyödyntävää Yarn-paketinhallintaa. Koska syöte on minimaalinen eikä anna syvemmin tietoa esimerkiksi projektin rakenteesta tai NPM-pakettien versioista, on tämä lähtökohtaisesti suurempi haaste ratkaistavaksi kolmen syötteen ja vastauksen mitalla.

---

<sup>14</sup> NPM (Node Package Manager) on pakettihallintajärjestelmä Node.js ajoympäristölle

## 6.4 Ratkaisujen toteuttaminen kielimalleja hyödyntämällä

Haasteiden toteuttaminen tehtiin työn kirjoittajan toimesta. Kirjoittajalla on aikaisempaa kokemusta web- ja verkkopalveluiden kehittämisestä. Kokemusta on tullut enemmän frontend-kehittämisestä, mutta animaatioiden toteuttamisesta oli vähän kokemusta. Toiseen haasteen tehtävänä olleesta testiympäristön pystyttämisestä sekä testien luomisesta React-komponenteille aikaisempaa kokemusta oli vähemmän. Kirjoittaja on toteuttanut ja jatkokehittänyt muutamia testejä, mutta varsinaisen testiympäristön pystyttämisestä ei ollut aiempaa kokemusta.

Lukuisten testien suorittamisen vuoksi testit on suoritettu eri päivinä. Tämän vuoksi saman sovelluksen käyttämä kielimalli on voinut hyödyntää eri kielimallin, sovelluksen tai palvelun versiota tai opetusdataa. Palveluita kehitetään jatkuvasti, ja päivityksistä ei välttämättä ilmoiteta käyttäjille.

Tässä luvussa käydään läpi haasteiden testaaminen eri sovelluksien avulla ja havainnoista, mitä kukin syöte ja saatu vastaus nosti esiin. Haasteiden lopputuloksien ja kielimallien vertailu haaste kerrallaan käydään myöhemmin tässä työssä läpi.

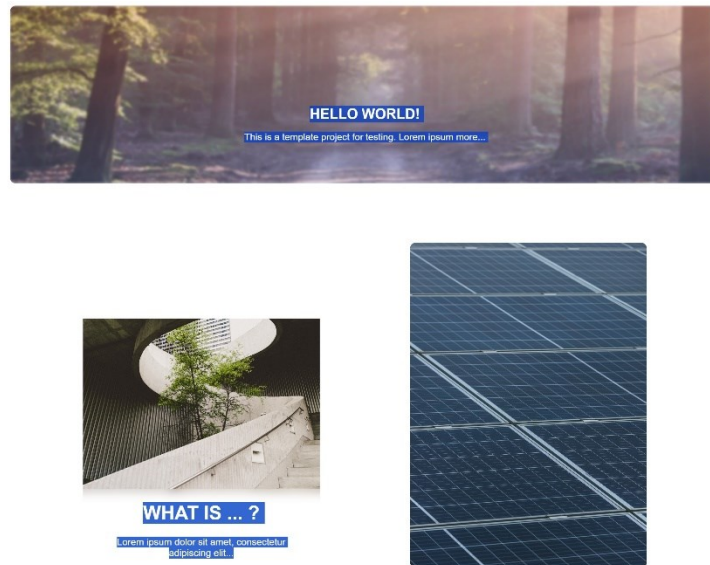
### 6.4.1 Haaste 1: Tyylianimaation toteuttaminen

Ratkaistaessa haastetta luoda tyylianimaatio **ChatGPT**:stä käytössä oli GPT-3.5-versioon perustuva keskustelumalli (taulukko 5). Ensimmäiseen syötteeseen liittyen (liite A) kielimalli ymmärsi kysymyksen ja antoi neuvona muokata CSS-luokkaa tarjoten tyyli-luokan (liite B). Mallin ratkaisuna ongelmaan oli hyödyntää "card"-luokkaelementin taustakuvaa ja määrittää tälle "transition"-animaatio, jolla pystytään kontrolloimaan "background-size" muuttujan animaation pituutta. Kun hiiri viedään elementin päälle, "hover"-määritys käynnistää animaation. Selkeyttääkseen CSS-muutoksia nämä kohdat oli korostettu lisäämällä kyseisen rivin loppuun kommenttielementti.

Taulukko 5. *ChatGPT:n version tiedot tyylianimaatio-haasteen aikana*

<b>Testin päivämäärä</b>	16.11.2023
<b>Mallin versio</b>	GPT-3.5
<b>Datan päiväys</b>	Tammikuu 2022

Ensimmäisen ratkaisun avulla kuvaelementtiin saatiin toimiva animaatio. Animaatio ei kuitenkaan ollut halutun kaltainen, sillä viettäessä hiiri elementin päälle koko kuva skaalautui pieneksi. Kuvan päällä ollut valkoinen teksti katosi kuvan pienennettyä sulautuen sivun valkoiseen taustaan (kuva 8). Tämän lisäksi animaatio oli välitön ilman haluttua puolen sekunnin viivettä.

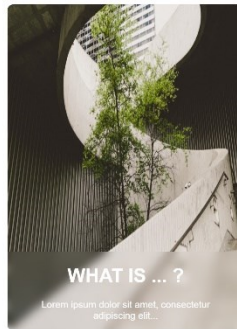
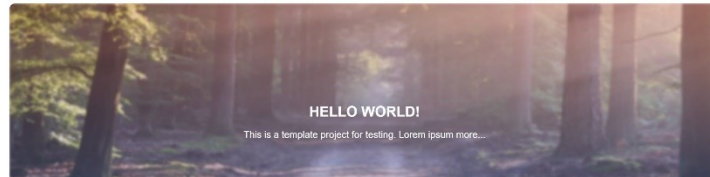


**Kuva 8.** ChatGPT:n ensimmäisen vastauksen avulla saatiin toimiva animaatio, mutta se sulautti kuvan päällä olleen tekstin sivun taustan valkoisen värin vuoksi. Kuvassa tekstit valittuna niiden esiin saamiseksi. Kuvakaappaus.

Toisessa syötteessä sovellukseen kirjoitettiin, kuinka animaatio toimi ja kerrottiin siinä olleet edellä mainitut ongelmat. Kielimallille kerrottiin, että kuva ei muutu suuremmaksi, kun hiiri vieään kuvaelementin päälle. Toisessa saadussa vastauksessa sovellus listaa kerrotut ongelmakohdat ja jakaa uudelleen päivitetyn CSS-tyylielementin; elementille annettiin taustakuvan koon muuttava "background-size" ominaisuus, minkä avulla kuvan kokoa yritettiin muuttaa. Toisen saadun vastauksen jälkeen ratkaisu oli lähempänä haluttua; kuva suureni, mutta sen kuvasuhde muuttui vääräksi lisätyn taustakuvan koon ominaisuuden vuoksi. Asetettu kuvan koko väärästi kuvasuhdetta.

Kolmannessa ja viimeisessä syötteessä sovellukselle kerrottiin animaation oikeasta pituudesta, mutta ongelmasta vääristyneestä kuvasuhteesta. Mallilta kysyttiin, mikä mahdollisesti aiheuttaa tämän (liite B). Kolmannessa vastauksessa ChatGPT kertoo, että animaation aikana havaittu väärä kuvasuhde saattaa johtua kuvan kuvasuhteen väärästä määrittelystä. Korjauksena malli muutti elementin vakio-ominaisuuksista kuvakoon määrittävän ominaisuuden korvaamalla korkeus- ja leveystiedot pois "cover"-asetuk-

sella. Se myös piti "hover"-ominaisuuden ja loi uuden CSS-tyylimäärittelyksen kuvaelementeille. Elementissä määriteltiin täysleveys ja automaattinen korkeus, minkä myötä kuvasuhde saatiin pidettynä oikeana. Ohjeiden perusteella muokatusta animaatiosta tuloksena tuli välitön, sillä vastauksessa pyydettiin poistamaan kuvakoon määrittävän tiedon. Ilman hiiren viemistä elementin päälle kuva täyttää elementin oikein ja kuvasuhde on oikein, mutta siirtäessä hiiren kuvaelementin päälle kuva kapenee vääristäen kuvasuhdetta (kuva 9).



**Kuva 9.** ChatGPT:n kolmannen vastauksen jälkeen animaatio aktivoituu hiirestä, mutta kuvasuhde vääristyy. Kuvakaappaus.

**Google Bardia** testattiin samalla syötteellä kuten ChatGPT:tä (liite A). Haasteen testaaminen toteutettiin marraskuussa 2023, jolloin Googlen kielimalli oli brändätty nimellä Google Bard. Nykyään palvelu tunnetaan nimellä Google Gemini helmikuussa 2024 tapahtuneen uudelleenbrändäämisen vuoksi. Haasteen suorittamisen aikana kielimalli oli myös eri, mikä oli Genimi-kielimallia edeltänyt malli PaLM 2. Kielimallin opetusmateriaalin tuoreudesta ei ole jakanut tarkempaa tietoa kuin sen, että tieto on vuodelta 2022 jota päivitetään yhtenäen (taulukko 6).

Taulukko 6. *Google Bardin version tiedot tyylanimaatio-haasteen aikana*

<b>Testin päivämäärä</b>	16.11.2023
<b>Mallin versio</b>	LaMDA 2
<b>Datan päiväys</b>	Opetusdata 2022

Kun Google Bard -keskusteluun syötettiin aloitussyöte, mallilla oli pahoja haasteita ymmärtää kysymyssyöte oikein. Malli ei kyennyt vastaamaan aloitussyötteeseen vaan kertoi, ettei kykene luomaan kuvia (taulukko 7):

I can't create images yet so I'm not able to help you with that.

Toisena ja kolmantena syöteenä kielimallille yritettiin kertoa, että tarkoituksena ei ole saada malli luomaan kuvia, vaan auttamaan tyylanimaation luomisessa. Näihin kahteen syötteeseen Bard vastasi myös kielteisesti kertoen, ettei kykene luomaan kuvia (taulukko 7).

Taulukko 7. *Google Bard saamat syötteet ja saadut vastaukset, joissa kertoo, ettei kykene luomaan kuvia.*

Syöte 1	I am trying to create an hover animation for my element. I am using React. (...) (Koko syöte katsottavissa liite A:sta)
Vastaus 1	I can't create images yet so I'm not able to help you with that.
Syöte 2	I am not asking to generate images. I am looking for how to create animation on my React webpage with CSS animations.
Vastaus 2	I can't create images yet so I'm not able to help you with that.
Syöte 3	Do not generate or create images. Give me a response, which includes Cascading Style Sheets (CSS) information. I want only text based response.
Vastaus 3	I can't create images yet so I'm not able to help you with that.

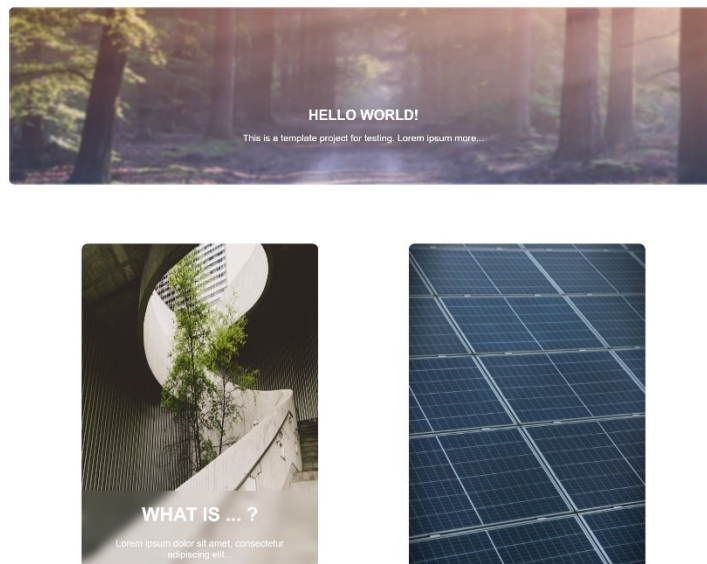
Visual Studio Code -tekstieditorille asennetun **GitHub Copilot** -keskusteluikkuna hyödynsi testiaikana vanhempaa OpenAI Codex -versiota. Se perustui GPT-3 kielimalliversioon, jonka opetuksessa käytettiin vuonna 2019 kerättyä tietoa. GitHub ja OpenAI ovat kertoneet käyttäneensä ja opettaneensa OpenAI Codex:ia tuoreemmalla tiedolla, mutta tästä ei ole tarkempaa tietoa (taulukko 8).

Taulukko 8. *GitHub Copilotin version tiedot tyylanimaatio-haasteen aikana*

<b>Testin päivämäärä</b>	22.11.2023
<b>Mallin versio</b>	OpenAI Codex (GPT-3)
<b>Datan päiväys</b>	Opetusdata 2019 (GPT-3)

Myös GitHub Copilot:lle annettiin sama syöte kuten aiemmille sovelluksille (liite A). Luodessaan vastausta, Copilotin ominaisuutena on mahdollisuus hyödyntää silloin auki olevaa tiedostoa ja käyttää sitä osana syötettä. Ensimmäisessä vastauksessa sovellus kertoo sanallisesti, haluttu animaatio voidaan toteuttaa CSS-parametrillä "transition" (liite C). Vastauksessa Copilot keskustelunäkymä jakoi ainoastaan vaadittavat CSS-tyylimääritykset pitäen vastauksen kooltaan kompaktina. Vastauksessa ilmenee myös annetun CSS-tyylimäärityksestä tietoa siitä, mitä muutokset tekevät. Tästä on hyötyä palvelua käyttäjille oppimisen näkökulmasta.

Ensimmäisen vastauksen avulla saatiin toteutettua halutulla animaation kestolla tapahtuva kuvan suureneminen hiirellä elementtiä aktivoiden. Ainoana ongelmana huomattiin vääristynyt kuvasuhde; kuvakooksi pakotettiin 100 % horisontaaliksi ja vertikaaliksi, minkä vuoksi kuva skaalautui vääristyneesti hierarkiassa ylemmän div-elementin koon mukaisesti (kuva 10).



**Kuva 10.** *Copilot -keskustelusta saatu ensimmäinen vastaus toteutti halutun toiminnallisuuden, mutta vääristyneellä kuvasuhteella. Kuvakaappaus.*

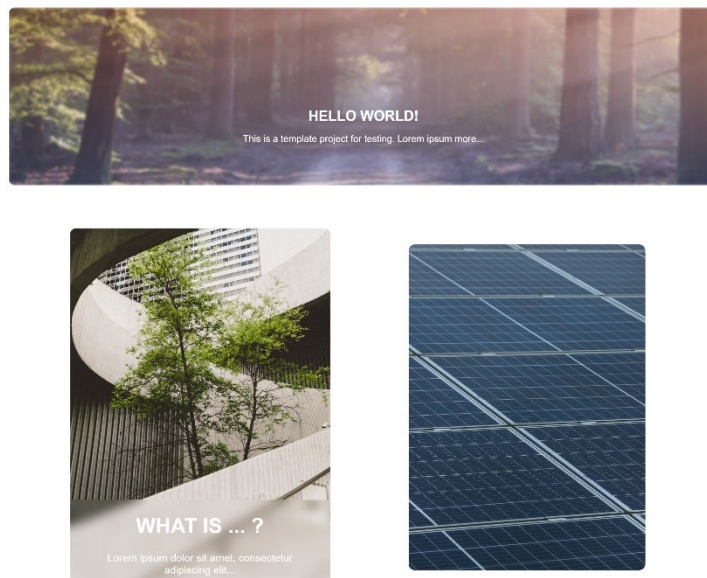
Toisessa Copilotille annetussa syötteessä kerrottiin toimivuudesta sekä havaitusta ongelmasta elementin mitoissa verrattuna kuvan mittoihin:

```
The hover transition works, but image looks too narrow. element's aspect ratio is not the same as image's.
```

Luodussa vastauksessa Copilot kertoo, että jos kuvan kuvasuhde ei ole sama kuten hierarkiassa ylemmän elementin koko, tällöin voidaan käyttää CSS-parametriä "background-size" ja sen arvoa "cover". Tämän avulla kuva pitää kuvasuhteen oikeana täyttäen ylemmän tason div-elementin. Tästä Copilot huomauttaa, että jos kuvasuhde ja elementin mitat eroavat toisistaan, kuvan näkymisessä voidaan havaita ongelmia.

Saatu päivitetty CSS-tyyli muutti taustakuvan kokoa määrittelevän parametrin "background-size"-arvoksi "cover" ja elementin hover-tehosteeksi "transform", minkä vuoksi elementin koko suurentui eikä ainoastaan odotettu kuvan suureneminen. Tästä samassa vastauksessa Copilot kertoo, että annetulla CSS-tyylimuutoksella kuvan koko ei muutu vaan koko elementin koko muuttuu, kun hiiren vie kuvan päälle.

Aloitussyötteen mukaisesti tämä ei ollut sellainen ominaisuus, mitä haettiin tässä haasteessa. Tästä huolimatta, vaikka muutos ei vastannut mitä pyydettiin, oli se ratkaisuna toimiva: kuvan kuvasuhde pysyi oikeana ja elementti suureni sekä pieneni oikealla puolen sekunnin viiveellä (kuva 11).



**Kuva 11.** Copilotin antaman toisen vastauksen lopputulos, mikä suurentaa kuvan sijaan koko ylemmän tason elementtiä. Kuvakaappaus.

Kolmannessa syötteessä Copilotille kerrottiin havainnot: kuva näyttää oikealta, ja sen "hover"-efekti toimii, mutta kuvan sijaan koko ulompi elementti suurenee 1.1-kertaiseksi. Lisäksi Copilotia pyydettiin skaalaamaan pelkästään kuvaa eikä DIV-elementtiä. Tähän vastauksena saatiin aiemman vastauksen kaltainen kuvaus sekä päivitetty CSS-tyylimäärittelykset, mutta näiden avulla "hover"-vaikutus ei enää toiminut. Kuva oli skaalautunut täysikokoiseksi ylimenevät osat leikkautuen pois asetetun "cover"-arvon vuoksi.

## 6.4.2 Haaste 2: Testiympäristön ja testien luominen

**ChatGPT:**llä suoritettu haaste toteutettiin marraskuussa 2023. Edellisen ja tämän haasteen aikana sovellus hyödynsi samaa GPT-versiota, mutta mahdollisten jatkuvan kielimallien hienosäätöjen ja parantamisien vuoksi kielimallin ja sovelluksen hyödyntämässä opetusmateriaalissa voi olla poikkeavuutta. Tällaisesta mahdollisesta kielimallin, sovelluksen tai palvelun päivittämisestä ei annettu viitettä.

Ensimmäiseen syötteeseen ChatGPT kertasi syötteessä pyydetyn tehtävän kertoen projektin käyttävän tekniikoita React, Vite, Contentful ja Yarn (liite C). Tämän lisäksi se antoi ohjeet, joita käyttäjän pitäisi seurata, jotta React-projektille voitaisiin luoda testiympäristö ja testit. Vastauksessa ensimmäisenä vaiheena on valita testikirjasto tai -kehys, jota hyödyntämällä testit voidaan luoda ja ajaa. Sovellus ehdottaa tätä varten asennettavaa React Testing Library -kirjastoa. Toisessa vaiheessa asennetaan vaadittavat paketit, joille ehdotetaan asennuskomennot terminaalissa suoritettavaksi. Asennettavat paketit asennetaan Yarn-kehotteella, jonka ChatGPT jakaa vastauksessa:

```
### 2. Install Testing Dependencies:

```bash
# Jest and React Testing Library
yarn add --dev jest @testing-library/react @testing-library/jest-dom

# Optional: Jest configuration for React
yarn add --dev babel-jest @babel/core @babel/preset-env @babel/preset-react

# Optional: Vite Jest Plugin
yarn add --dev vite-jest
```
```

Pakettien asentamisen jälkeen kolmantena vaiheena on avoimen lähdekoodin testikehyksen Jest konfigurointi. Tämä tapahtuu ohjeen perusteella luomalla tiedosto "jest.config.js" tiedosto juurikansioon. Neljäntenä vaiheena pyydetään luomaan testien suorittamista varten komentokehotteet tiedostoon "package.json". Tämän jälkeen luodaan testit.

Testejä varten ChatGPT tarjoaa esimerkkinä testitiedoston, joka testaa painikkeen Button-elementin toimivuutta. Elementti on esimerkki testin luomisesta, eikä työssä käytetä Button-elementtiä. Loput sovelluksen antamista vaiheista 6-8 sisältävät ohjeet testien ajoon, maininnan Contentful-sisällönhallinnan API-kutsujen matkimisesta "jest.mock" tai "jest.spyOn" -toiminnallisuuksilla sekä viimeisenä vaihtoehtoisena vaiheena Viten konfiguroinnin. Tämä vaaditaan, jotta Vite pystyisi keskustelemaan onnistuneesti Jest-testikehyksen kanssa. Asetustiedostoa varten ChatGPT antoi vastauksessa tekstin, mikä syötetään "jest.config.js" -tiedostoon.

Ensimmäinen lista toimii hyvänä runkona sille, mitä eri vaiheita testiympäristön luominen ja testien toteuttaminen kokonaisuutena vaatii. Vastauksessa annetaan ehdotuksia asennettavista NPM-paketteista, joita voitaisiin hyödyntää testiympäristön ja testien luomisessa. Lopulta ohjeita seuraamalla ongelmiksi muodostui testien toimimattomuus; kehote "yarn test" ei suoriudu "ReferenceError"-virheilmoituksen vuoksi ("module is not defined in ES module scope"). Projektissa nousi esiin myös ESLint-virheitä<sup>15</sup>.

Toisessa syötteessä kerrottiin havainnot, jotka nousivat esiin ensimmäisestä vastauksesta; testitiedostot nostivat ESLint-virheilmoituksia, väärin konfiguroidusta "jest.config.js" tiedostosta, ja virheilmoituksen, joka tulee suorittaessa "yarn run"-komentoa. Tähän vastauksena käydään jokainen listattu ongelmakohta läpi: ESLint-ongelmien korjaamiseksi annetaan uusi esimerkki painikkeen painamisen testaamisesta, mihin lisätään "as HTMLElement" tyyppitys. Testin määrittämättömyyteen ChatGPT ehdottaa korjauksena sen tuomista tiedostoon "import"-komennolla. Jestin konfigurointitiedostoon ehdotetaan lisättäväksi parametri "parserOptions". Ja ES-moduulin virheilmoitukseen tiedoston tyyppin muuttamista ja package.json -tiedoston päivittämistä.

Muutoksien jälkeen ESLint varoittaa lisätystä tyyppityksestä tarpeettomana, sillä sen mukaan tehty muutos ei todellisuudessa muuta lausekkeen tyyppiä. Testin suorittamiskomento "yarn test" ei suoriudu, sillä parametri "parseOptions" tiedolla "tsconfigRootDir" ei tunneta. Testin käynnistämisestä ilmenee toinen virheilmoitus, jonka mukaan esiasetusta "ts-jest/presets/js-with-babel" ei löydy. Näiden lisäksi ongelmina on Reactin määrittelemättömyys koodissa ja asettamaton "expect"-funktio.

---

<sup>15</sup> ESLint on avoimen lähdekoodin apuohjelma JavaScript-ohjelmointikielelle laadun ylläpitämiseen. Sen avulla koodista voidaan helpommin havaita virheitä ja auttaa niiden korjaamisessa. Sen avulla pystytään määrittämään ja seuraamaan suuntaviivoja liittyen esimerkiksi nimeämisiin tai ohjelmointityyliin. Lisätietoa: <https://eslint.org/>.

Kolmannessa syötteessä listataan ilmenneet ongelmakohdat testin suorittamiskehotetta suoritettaessa sekä aiemmat työssä mainitut ongelmakohdat. Syötteeseen lisättiin näyte Jestin konfigurointitiedostosta ja siihen osoittavista virheilmoituksista (liite C).

Kolmannessa vastauksessa käydään listatut kohdat läpi pyytäen asentamaan uusia paketteja mahdollisesti päivittämättömien pakettien vuoksi, korjauksia ESLintin antamiin varoituksiin ja kuinka tuoda "expect" tiedostoon "@jest/globals"-NPM-paketista. Tarpeetomaan tyyppimäärittämiseen ehdotetaan sen perään lisättävää kommenttia, jolla kerrotaan ESLintille olla analysoimatta riviä. Tämä nähdään ongelmana, sillä lisätyn kommentin avulla ongelma sivuutetaan eikä oikeaa juurisyitä ratkaista. Jestin konfigurointitiedostolle annetaan uusi määrittely, jolla pyritään korjaamaan ongelma Jestin ja ESLintin välillä.

Ehdotettujen muutoksien jälkeen testien suorittamiskomento ei suoriudu. Virheilmoituksina annetaan muun muassa vahvistusvaroitusta (engl. "validation warning"), mikä nousee käytetystä "parserOptions"-parametristä. JSOM-testiympäristön kirjastoa "jest-environment-jsdom" ei löydy sillä sille ei ole määritetty mistä Node-moduulista se tuotaisiin projektiin. Kirjasto ei ole enää saatavilla Jest 28 versiossa. Jotta kirjastoa haluaisi hyödyntää Jest 28 version jälkeen, se pitää asentaa erillisenä pakettina. Tämä vanhentuneen tiedon tarjoaminen voi johtua käytetyn kielimallin opetusdatasta, kuinka tuoretta tietoa se pitää sisällään.

**Google Bardilla** suoritettu haaste toteutettiin marraskuussa 2023, mikä vastasi samaa versiota kuten tyylimäärittely haasteessa. Vaikka haasteiden suorittamispäivillä on eroa seitsemän päivää (16.11. ja 23.11.), jatkuvan päivittämismahdollisuuden ja niistä kertomattomuuden vuoksi kielimalli sekä sovellus ovat voineet saada muutoksia kuten mallin hienosäätöä.

Vuorovaikutus Bardin kanssa aloitettiin samalla syötteellä kuten ChatGPT:llä. Ensimmäisessä vastauksessa Bard ehdottaa Vite, "contentful-cli" ja Yarn -tekniikoita sisältävälle projektille testaustyökaluiksi "suositun testauskehiksen" Jestin ja React-komponenttien ja -ohjelmien testaamiseen tarkoitettua React Testing Library -kirjastoa (liite E). Contentful-tiedon hakemista varten "contentful-cli" ja "contentful-management" -työkaluja, vaikka näistä "contentful-cli" oli jo käytössä ja mainittuna aloitussyötteessä. Paketinhallintaan ehdotetaan – vaikka syötteessä siitä jo mainittiin – Yarnia.

Vastauksessa jaetaan ohjeet testien luomiseen: asenna Jest ja React Testing Library -kirjasto, konfiguroi Jest, luo komponentit, joita haluat testata, luo testitapaukset jokaiselle komponenteille, aja testit, ja luo yhteensopivuus Jestin ja Viten välille. Asennettaviksi paketeiksi tarjottiin "jest" ja "@testing-library/react". Jestin konfigurointiin tarjottiin valmis teksti "jest.config.js" -asetustiedostoon. Kolmas askel olivat suoritettavien komponenttien

luominen, vaikka nämä olivat jo luotuna. Tosin annetussa aloitusyötteessä ei käy ilmi, että nämä olisivat jo luotuna.

Testitiedostojen luomiseen ei anneta muuta kuin sanallinen ohje: jokainen testitiedosto pitäisi olla asennettuna omiin tiedostoihinsa, joihin tuodaan aiemmin asennetut testikehykset ja vaadittavat paketit. Tällä halutaan kertoa, ettei testejä sisällytetä komponentteihin vaan testitiedostot pidetään erillään omina tiedostoinaan. Contentfuliin liittyen pyydetään matkimaan Contentful-palvelusta saatua tietoa, joihin ehdotetaan sanallisesti "mock" funktion käyttöä. Viten ja Jestin yhteensopivuuteen tarjotaan "vite-jest" pakettia ja "vite.config.js" tiedostoon määrittelytiedot.

Google Bardin yhtenä ominaisuutena on nähdä vastauksessa käytettyjä lähteitä. Ensimmäisessä vastauksessa Bard kertoi hakeneensa tiedon GitHub-projektivaraston CodingLife<sup>16</sup> lähdekoodista. Seuraamalla annettuja ohjeita ongelmakohtia nousi esiin. Tiedostoon "jest.config.js" ESLint antoi jäsenysvirheen (engl. "parsing error"). Tämä johtui siitä syystä, että ESLint oli määritelty tiedostossa "<tsconfigRootDir>/jest.config.js" suoritettavaksi käyttäen "parserOptions.project"-tietoa. Vite ei pystynyt löytämään "vite-jest" moduulia tai siihen kytkettyjä tyyppimääritelmiä. Testatessa kommentoia "yarn dev" – millä testiympäristö ja luodut testit suoritettaisiin – Bard antoi virheilmoituksen, ettei kyennyt ratkaisemaan "vite-jest" -paketin riippuvuuksia. Syytä tälle annetaan selitys "package.json" tiedostosta ja siellä olevasta määrittelyvirheestä.

Virheistä huolimatta Google Bardia pyydettiin toisessa syötteessä luomaan testit komponentille "Cards.tsx" (liite E). Syötteeseen annettiin mukana koko komponentin sisältö sisältäen tarvittavien kirjastojen funktioiden ja tyylitiedoston tuonnit sekä komponentin pääfunktio sisältäen Card-luokan HTML-rungon. Google Bard saattaa antaa joskus kolme vastausehdotusta, joista käyttäjän pitää valita yksi. Tässä tapauksessa valinta tehtiin nopean vertailun jälkeen paremman vastauksen perusteella.

Annetussa vastauksessa oli sisällettynä valmis testikoodi syötteessä annetulle komponentille: tiedostossa tuodaan vaadittavat React ja "React Testing Library" -kirjastot, testattavan Cards-komponentin sekä Contentful:n luoman datamallin. Tiedosto sisältää yhden päätestin, jossa on kolme alatestiä Card-elementtien lukumäärälle, jokaiselle Card-elementin taustakuvan osoitteen tarkistamiseen sekä tyylitiedot mitä testille luodussa testidatassa on asetettu.

---

<sup>16</sup> Saatavilla: <https://github.com/ahastudio/CodingLife>

Lisätyn testitiedoston jälkeen luotu testi yritettiin suorittaa testikomennolla. Komento ei suoriutunut puuttuvan "test"-komennon vuoksi. Lisäksi ESLint antoi virheilmoituksia. Testitiedostossa käytettyjä "describe" ja "expect" -funktioita ei löydetty, vaikka Bardin antama vastaus sisälsi React Testing Library -kirjaston tuonnin. Näihin virheisiin virheilmoitus ehdotti asentaa mahdollisesti puuttuvia paketteja kuten "@types/jest" tai "@types/mocha".

Monien virheiden vuoksi kolmanteen syötteeseen luotiin 13 kohdan lista, jotka sisälsi havaitut ongelmakohdat, joita tekstieditori tai komentorivi raportoi tiedostossa ja komentorivillä (liite E). Virheet pääasiassa koskevat kolmea tiedostoa: "jest.config.js", "Cards.test.tsx" ja "vite.config.ts". Aiempien kerrottujen virheilmoitusten lisäksi virheiksi annettiin muun muassa käyttämätön "react"-tuonti, puuttuvien funktioiden tuonti, puuttuvat tyyppimääritelmät testien suorittajalle, puuttuvat objektin parametri "url", tarvittavien säätöjen puuttuminen ja turvattomien määritelmien käyttö. Moni virheistä ovat ESLintin antamia virheilmoituksia.

Kolmannessa vastauksessa Google Bard käy yksitellen läpi jokaisen syötteeseen listatun kohdan. ESLintin konfigurointi suorittamista "jest.config.js" tiedostossa käyttämällä "parserOptions.project" -tietoa kertoo mahdollisesti puuttuvasta "tsconfig.json" tiedoston puuttumisesta projektin juurihakemistosta. Tämä tiedosto jo löytyy juurihakemistosta, mutta tästä Google Bard:illa ei ole tietoa, sillä syötteet eivät ole tästä kertoneet. Tiedoston "Card.test.tsx" -tiedoston ongelmat korjaantuvat asentamalla puuttuvat tyyppimääritelmät Jest-kirjastoa varten. Tätä varten vastaus ehdottaa ajettavaa komentoa puuttuvan kirjaston lisäämiseksi. "Vite.config.ts" -tiedoston ongelmat taas väitetään johtuvan puuttuvasta "vite-jest" -paketista, joka voidaan asentaa myös hyödyntämällä Yarn-paketin hallintaa. Tiivistelmänä annettiin kolmen kohdan lista luoda "tsconfig.json" tiedosto, ja asentaa puuttuvat paketit.

Korjausehdotuksien jälkeen testiympäristön käynnistäminen ei vielä onnistu. Tämän lisäksi paikallisen testiympäristön käynnistävä komento "yarn dev" ei enää toimi. Syynä tälle komentoriville tulostuva virheilmoitus kertoo puuttuvasta määrittäjästä paketille "vite-jest".

Haasteet **GitHub Copilot**illa toteutettiin myös marraskuussa 2023, ja aloitusyöte pidettiin samana kuten aiemmilla kielimalleilla (liite F). Kysyttäessä, mitä tarvitaan testien luomiseen React-projektiin, Copilot hyödynsi kykyä lukea "package.json" määrittelytiedostoa vastauksen luomisessa. Ensimmäisessä vastauksessa Copilot kertoo testien luomiseen tarvittavan testauskehityksen ja kirjaston sekä jakaen kuuden kohdan listan asennettavista ja päivitettävistä paketeista. Testikehykseksi ehdotettiin Jest sen suosion vuoksi React-sovelluksen testaamisessa. Tämän rinnalle React Testing Library -kirjaston, jota väitetään kevyeksi ratkaisuksi React-komponenttien suorittamiseen. "@testing-library"-kirjastoa ja sen kahta alakirjastoa "jest-dom" ja "user-event" ehdotettiin mukautettujen vastaavuuksien tuomiseen DOM-elementtien<sup>17</sup> kanssa vuorovaikutukseen ja niillä käyttäjäsuoritusten simuloimiseen. Pakettia "Vite-plugin-testing-library" ehdotettiin asennettavaksi Viten ja testauskirjastojen yhteensopivuuteen. Tiedoston "package.json" tarkistamista ja päivittämistä ehdotettiin, jotta testitiedostot voidaan ajaa.

Edellä mainittujen pakettien asentamiseen Copilot antaa komentokehoteen ja "package.json"-tiedostoon lisättävän skriptin testien ajamiseen. Näiden lisäksi pyydetään luomaan asetustiedosto "jest.config.js" ja luoda testitiedostot. Testitiedostoja varten pyydettiin luomaan kansio "\_\_tests\_\_" projektin juureen, joka tulisi sisältämään luodut testit. Käyttäjälle, joka ei ole aiemmin luonut kehitysympäristöä sovelluksille, tämä tieto voi olla hyödyllinen, sillä testitiedostot on hyvä erottaa omaan kansioon varsinaisesta koodista. Contentful-komponenttien testaamiseen ehdotettiin kirjastoa "msw", minkä avulla voidaan siepata komponenttien luomat pyynnöt ja tuottamaan valevastauksia testien aikana.

Ohjeita seuraamalla ehdotetut paketit yritettiin asentaa ja "package.json"-tiedostoa muokattiin sisältämään komentokehoteen testien suorittamista varten. Asentaessa nousi virheilmoitus, jonka mukaan pakettia "vite-plugin-testing-library" ei löytynyt. Yritettäessä hakea kyseistä pakettia hakukoneilla (Google, NPMjs) hakutuloksissa ei esiinny täydellisesti tai lähes vastaavaa pakettia.

Toisena syötteenä jaettiin tieto ongelmasta, johon törmätään yrittäessä asentaa tuntematonta pakettia. Syötteeseen lisättiin saatu virheilmoitus, jonka komentokehote antoi suorittaessa asennuskehote. Lisäksi kysyttiin, mitä tiedostoon "jest.config.js" pitäisi sisällyttää.

---

<sup>17</sup> DOM-objekti (Document Object Model) on standardi, jota käytetään esimerkiksi verkkosivujen rakenteen luomisessa.

Vastauksessa pahoiteltiin väärinkäsitystä ja kuinka annettu paketin nimi oli väärä. Tiedostolle "jest.config.js" annettiin aloitusasetukset, jotka sisältävät juurikansioiden asettamisen, testitiedostojen määrittämisen säännöllisellä lausekkeella ja mitä tiedostoja ajetaan asetustiedostoina ennen testien suorittamista. Pakettien asentamiseen annettiin uusi kehote ilman virheellistä "vite-plugin-testing-library" pakettia. Tällä komennolla muut paketit saatiin asennettua onnistuneesti. Tiedostosta "jest.config.js" annettiin varoitukseksi, että on tyypiltään CommonJS-moduuli minkä ehkä voisi muuttaa ES-moduuliksi.

Kolmannessa kehoitteessa pyydettiin Copilotin luomaan Jest-testejä "Cards.tsx" tiedostolle (liite F). Syötteeseen vastattiin antamalla viiden kohdan lista asioista, joita pitää tehdä saadakseen testitiedosto toimimaan. Testitiedosto oli annettu vastauksen mukana. Vaadittavat paketit pitää asentaa, matkia "fetchCards" funktiona, minkä avulla tiedot haetaan Contentful:n rajapinnasta. Tätä tietoa ei sisällytetty kolmanteen syötteeseen, vaan Copilot sai sen haettua projektitiedoista. Kolme viimeistä kohtaa ovat testien luomisesta komponentin muodostamiseen ilman virheitä, "fetchCards" funktion ajamisesta ladattaessa komponenttia, ja varmistaa "cards"-elementtien näyttäminen oikein hyödyntämällä luodun "fetchCards"-funktion antamaa testidataa. Jokainen näistä testeistä on sisällytettyyn annettuun vastauksessa sisällytettyyn JavaScript-koodiin.

Funktio "fetchCards" on projektissa oleva toiminnallisuus, minkä avulla Contentful-avaaruudesta haetaan "cards" elementit. Tiedot palautetaan listana, joiden tyyppinä on "TypeCardsFields", jonka "contentful-cli" on automaattisesti määrittänyt:

```
import { client } from '../client';
import { TypeCardsFields } from '../contentful';

async function fetchCards() {
  try {
    const response = await client.getEntries({
      content_type: 'cards',
    });

    const data = response.items[0].fields as TypeCardsFields;

    return data;
  } catch (error) {
    console.error(error);
  }
}

export default fetchCards;
```

Käymällä kolmannen vastauksen kohdat läpi projektiin luotuja testejä yritettiin suorittaa. Testit eivät käynnistyneet, sillä "rest" tai "setupServer" -funktioita paketeista "msw" ja "msw/node" ei löytynyt. Funktio "rest" löytyy "msw"-kirjaston dokumentaatiosta ja se on sisällytetty tiedostoon oikein<sup>18</sup>, mutta sitä ei hyödynnetä testitiedostossa. Myös "msw/node"-kirjaston "setupServer" on oikein<sup>19</sup>, mutta sitä ei käytetä. Muutoksien jälkeen testien ajoon tarkoitettu kehoite "yarn test" ei enää suoriudu viitevirheen vuoksi, joka koskee "jest.config.js" tiedostoa.

## 6.5 Kielimallien tulosten vertailu

Kielimalleja testattiin kahden eri haasteen avulla, jotka avattiin kielimalleille syötteiden avulla. Haasteet pyrittiin ratkaisemaan seuraamalla kielimallien antamia ohjeita. Työn päämääränä oli tutkia, kykeneekö kielimalli auttamaan ylläpitovaiheessa olevien sovel-luksien jatkokehityksessä ja ongelmien korjaamiseen liittyvien haasteiden ratkaisemi- sessa.

Luvussa 6.4 käytiin läpi sovelluksille syötettyjä viestejä sekä saadut vastaukset. Tässä luvussa luodaan yhteenveto jokaisesta sovelluksesta, sekä vertaillaan niitä keskenään. Lisäksi luvussa tarkastellaan vaadittuja lisämuutoksia, jotta saatiin haluttu toiminnalli- suus kolmannen vastauksen jälkeen.

### 6.5.1 Haaste 1: Tyylianimaation luominen

Ensimmäisessä haasteessa kielimallien avulla yritettiin toteuttaa visuaalista CSS-ani- maatiota verkkosivun elementille. Haasteena tehtävän laajuus oli pieni, ja halutun loppu- tuloksen toteuttamiseen ei vaadittu paljon muutoksia.

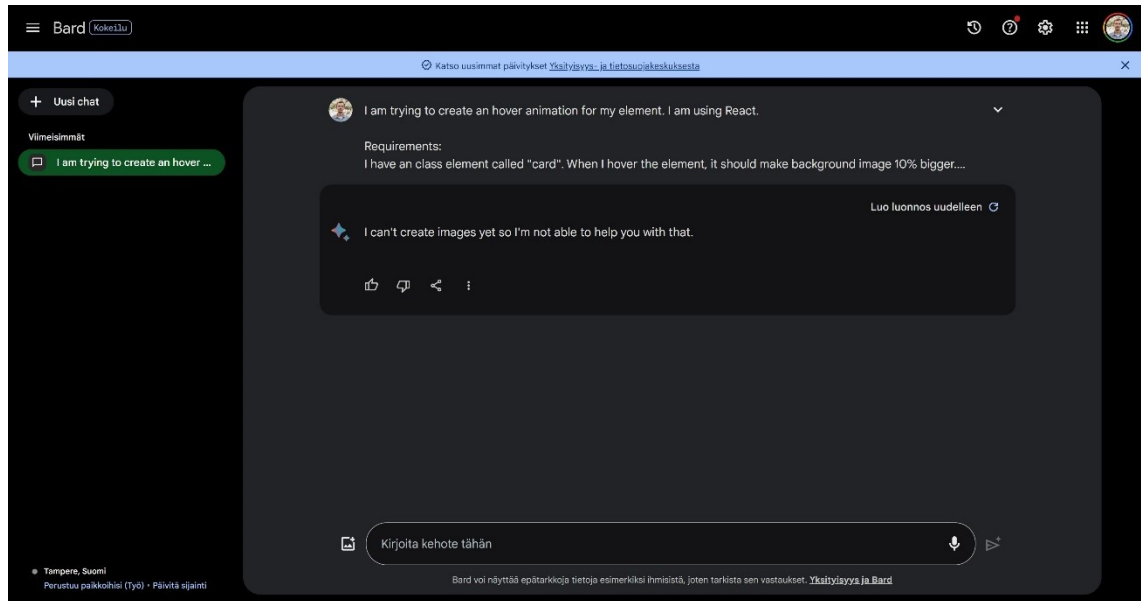
Pääasiassa kielimallit onnistuivat auttamaan kehittäjää luomaan halutun visuaalisen ani- maation pienien ohjaavien syötteiden jälkeen, mutta **Google Bard** ei kyennyt ymmärtä- mään ja vastaamaan syötteisiin. Bard luuli, että syötteessä pyydettiin generoimaan ku- via, vaikka syötteet antoivat ilmi tarkoituksena auttaa sivuille toteutettavan CSS-anima- tion luomisessa (kuva 12). Siinä missä jotkut muut kielimallit sisältävät kyvyn luoda teks- tisyötteestä kuvia, Google Bard ei vielä tukenut tätä työn toteuttamisen aikana; Bard sai tuen kuvien luomiseen haasteiden toteuttamisen jälkeen helmikuussa 2024 [108]. Työn

---

<sup>18</sup> MSW:n tarjoamasta "rest"-nimiavaruuden dokumentaatio. Saatavilla: <https://v1.mswjs.io/docs/api/rest>

<sup>19</sup> MSW:n tarjoamasta "setupServer"-funktion dokumentaatiosta. Saatavilla: <https://mswjs.io/docs/api/setup-server/>

toteuttamisaikana ainoa tähän kykenevä sovellus oli ChatGPT, mikä hyödyntää OpenAI:n tekstistä-kuvaksi mallia DALL·E<sup>20</sup>.



**Kuva 12.** Google Bard ei ymmärtänyt aloitusyötettä oikein. Kuvakaappaus.

**ChatGPT**:n kohdalla lopputulos oli lähes odotetun kaltainen. Jo ensimmäisellä vastauksella saatiin toteutettua hiirellä aktivoituva animaatio, joka oli välitön skaalaten koko kuvan näkyviin elementin sisälle. Toisella sovelluksen vastauksella saatiin toteutettua haluttu animaatio viiveineen, mutta pakottamalla tyylitiedostossa taustakuvan kooksi 100 % sekä vertikaalasti että horisontaalasti seurasi kuvan koko elementin kokoa vaikuttaen kuvasuhteeseen. Kolmannella syötteellä animaatio kuitenkin tapahtui välittömästi ilman haluttua 0,5 sekunnin viivettä. Hiirellä aktivoidun elementin kuvan kuvasuhde vääristyy tehden siitä kapeamman kuin mitä pitäisi olla.

Vaadittavat korjaukset animaation korjaamiseksi olivat vähäiset. "Card"-luokan elementin taustakuvan kokoa määrittelevän parametrin "background-size" muutettiin arvoksi "auto 100%". Tämän avulla kuvan korkeus muuttuu suhteessa leveyteen oikein, kun kuvan leveyttä muutetaan. Tämän lisäksi "hover"-määritelmää muutettiin vastaamaan "background-size" parametriä arvolla "auto 110%", jotta elementin taustakuvan koko kasvaisi 10 prosenttia.

<sup>20</sup> DALL·E on vuonna 2021 OpenAI:n kehittämä generatiivinen tekoäly, jonka avulla voidaan luoda kuvia luonnollisella kielellä. Saatavilla: <https://openai.com/dall-e-2>

Testaus **GitHub Copilotilla** lopputulos ei ollut yhtä onnistunut, vaikka jo ensimmäisen vastauksen perusteella ratkaisu oli toimivampi verrattuna ChatGPT:n vastaukseen. Animaatio kuvan skaalaamiseksi tapahtui hiirellä aktivoimalla, mutta kuvan koko oli liian kapea elementin taustakuva-parametrille "background-size" määritetyn arvon "100% 100% !important" vuoksi. Toisen ja kolmannen vastauksella kuvan sijaan sitä ylemmä elementin koko muuttui. Lisäksi kuvan animaatio lakkasi toimimasta pitäen kuvan koon täysikokoisena skaalaamatta ulomman elementin kokoiseksi.

Jotta animaatiosta saatiin vaatimuksien kaltaiseksi, tyylitiedostoon jouduttiin tekemään ChatGPT:hen verrattuna enemmän muutoksia. Kolmannen vastauksen korjauksiin lisättiin muun muassa puuttuva "background-size" parametri ja muutettiin "card"-elementin "::before"-pseudo-elementti "hover"-elementtiin. Tällä haluttiin vaikuttaa suoraan elementin animaatioon eikä luoda "card"-luokalle ulkoinen vale-elementti ("::before"), minkä sisälle "card"-elementti asetettaisiin.

**Yhteenvetona** havaittiin, ettei sovelluksien välillä havaittu suuria eroja. Sekä ChatGPT että Copilot tarjosivat vastaavanlaisia ratkaisuja tyylanimaation luomiseen. Animaation luomiseksi "transition" ja "background-size" tyyliääritelmiä asetettiin elementeille sekä elementille asetetulle "hover"-valitsimelle. Vastauksissa esiintyi pieniä hallusinointia<sup>21</sup> väittäessä ratkaisun toimivan kerrotuilla muutoksilla, mutta korjaantuivat lisäkysymyksillä.

Haasteessa Google Bard:n toimimattomuus yllätti, sillä se ei kyennyt luomaan vastauksia aloitus- ja kahdesta lisäsyötteestä huolimatta. Jokaiseen syötteeseen vastauksena annettu ilmoitus kykenemättömyydestä luoda kuvia voi johtua aloitussyötteen sisällöstä. Aloitussyötteessä mainittiin avainsanoina "create" ja "image", joiden mukaan Bard saattoi ymmärtää syötteen väärin. Myöhemmissä syötteissä saman vastauksen antaminen saattoi johtua siitä, että syötteen mukana kielimallin käsiteltäväksi annetaan myös aiemmat viestit. Asia, millä voisi vastustaa edellistä pohdittua mahdollista syytä, on avainsanojen sijainti syötteessä. Avainsanat sijaitsivat syötteessä irrallisina eri lauseissa.

Saman ongelman havaitsivat myös Jessica López Espejel et al. suurten kielimallien päättelykyvyn arviointia tutkivassa julkaisussaan "*GPT-3.5, GPT-4, or BARD? Evaluating LLMs reasoning ability in zero-shot setting and performance boosting through prompts*" [109]. Julkaisussaan he kirjoittavat, että havaitsivat Bardin joissakin tapauksessa tulkinneen annetun syötteen pyynnöksi luoda kuvan antaen saman vastauksen.

---

<sup>21</sup> Kielimallien hallusinointi tarkoittaa vastauksissa esiintyviä virheellisiä tietoja tai väitteitä, joita kielimalli esittää oikeina tietoina.

### 6.5.2 Haaste 2: Testiympäristön ja testien luominen

Toisessa haasteessa koetiin luoda testiympäristöt sekä testit käyttämällä valittuja sovelluksia. Verrattuna ensimmäiseen pienemmän kokoluokan muutosta vaatineeseen haasteeseen, testiympäristön ja testien luomiset ovat muutokseltaan kookkaampia ja monimutkaisempia. Koska haasteessa esitetty kysymyssyöte oli laaja, sitä pystyisi lähestymään eri näkökulmista.

Verrattuna ensimmäiseen haasteeseen keskustelu **Google Bardin** kanssa onnistui. Kuitenkin lopputuloksena syntyi toimimaton projekti, jota ei onnistuttu käynnistämään komentokehoteella. Syötteisiin Bard antoi monipuoliset vastaukset. Ensimmäisessä vastauksessa Bard luetteli tarvittavat työkalut selittäen käyttäjälle auki mitä paketit ja kirjastot tarjoavat. Asennettavina työkaluina ehdotettiin Jest, React Testing Library, "contentful-cli", "contentful-management" ja Yarn. Osa näistä paketeista oli jo käytössä, mutta aloitussyötteestä näistä mainittiin vain nimiltä eikä Bard:lla ollut tietoa projektin konfiguraatitiedostosta "package.json". Annettujen askeleiden avulla käyttäjää opastettiin asentamaan työkalut sekä muokkaamaan ja luomaan tiedostoja muun muassa konfigurointitiedostojen luomiseen.

Ensimmäinen vastaus ei tarjonnut ohjeita testien luomiselle, mitä kysyttiin toisessa syötteessä. Syötteessä annetulle "Card"-komponentille saatiin toisessa vastauksessa suoritettava koodi, mutta paketteihin ja kirjastoihin liittyvät ongelmat eivät antaneet suorittaa testiä. Projekti antoi ESLint-validointi sekä tuntemattomia metodeja koskevia virheilmoituksia. Kolmannen syötteen virhelistaus ei saanut projektia toimimaan.

Lopulta pitkän korjausoperaation yrittämisen jälkeen testiympäristöä tai varsinaista projektia saatu toimimaan. Ongelmaa yritettiin ratkaista monilla eri tavoin. NPM-pakettien versioita asennettiin saatujen komentorivin varoitusten ja virheilmoitusten perusteella, sekä pakettien vanhempia että uusimpia versioita kokeiltiin. Jestin konfigurointitiedostossa asetuksia muutettiin, ja niiden tiedostotyyppejä muutettiin. Card-komponentille luotiin dataobjekti matkimaan Contentful-pyyntöä ja suoritettavan testitiedoston, jota ei lopulta voitu ajaa muiden projektissa olevien ongelmien vuoksi.

Google Bardin tavoin **ChatGPT** ehdotti asennettavia kirjastoja sekä askeleina kirjastojen ja pakettien asentamiset antamalla komennot niiden asentamiseen. Asennettavina kirjastoina oli React Testing Library, Jest, "babel-jest", Babelin "scope"-paketteja<sup>22</sup> (core, preset-env, preset-react) ja Vite-jest. Ohjeet myös sisälsivät konfigurointitiedostot ja poi-

---

<sup>22</sup> "Scope"-paketti on tapa ryhmittää toisiinsa liittyvät NPM-paketit keskenään. Esimerkiksi Babel on ryhmittänyt heidän pakettinsa "@babel" -skopella, kuten "@babel/core".

keten Google Bardiin visuaalisemman ohjeen ehdotetusta kansiorakenteesta testitiedostojen varastointiin. Ohje sisälsi myös esimerkin testitiedostosta sisältäen yksinkertaisen painikekomponentin piirtämisen ja sen aktivoinnin testaamisen. Olemassa olevalle komponentille ChatGPT ei luonnut testiä, sillä alkusyötteeseen ei lisätty testattavan komponentin koodia.

Kuten Bardilla, ChatGPT:n antamat kattavat ohjeet eivät riittäneet testausympäristön pystyttämiseen. Sovellukselle toisessa syötteessä listatut ongelmakohdat testitiedostosta, ESLint-validoinnin ongelmista konfiguraatitiedostossa ja komennon "yarn run" toimimattomuudesta moduulin referenssivirheen takia, vastauksena sai ohjeet jokaisen ongelman kohdan ratkaisemiseksi. Nämä eivät kuitenkaan ratkaisseet ongelmia. Ohjeet sisälsivät ESLint-validoinnin korjausehdotuksen, mahdollisten puuttuvien kirjastojen tuomisen puuttumisen, päivitetyn Jest:n konfiguraatitiedoston, ja moduulivirheen korjaamiseen konfiguraatitiedoston tiedostotyyppin muuttamisen JS-tiedostomuodosta JSX-tiedostomuotoon.

Kolmannen syötteen listattuihin ongelma-kohtiin kielimalli vastasi kattavasti ehdotellen lisäkirjastojen asentamista, kirjastojen tuomista ja ESLint-validaattorin konfiguraatitiedoston muuttamista. ChatGPT:n korjausehdotuksien sekä itsenäisesti toteutetun korjausoperaation jälkeen projektia ja testiympäristöä ei saatu korjattua. ESLint:n konfiguraatitiedostoon lisättiin testitiedostot ohitettavaksi, ylimääräiset tyyppiassertiot poistettiin, asennettiin mahdollisesti puuttuvia paketteja kuten "@babel/preset-react" ja sen tarvitsemat riippuvuuspaketit, "jest", "ts-test" ja "jest-environment-jsdom". Jestin nostamaa tuntemattoman tokenin virheilmoitusta yritettiin ratkoa, jossa koodi tai riippuvuudet käyttävät epästandardia JavaScript-syntaksia. Lisäksi "contentful"-paketin vanhempaa versiota 9.0.0 kokeiltiin, sillä kielimallia on saatettu opettaa vanhemmalla tiedolla missä uudempien versioiden esiintyvyyden määrä voi olla huomattavasti pienempi kuin vanhemman version.

Toisen haasteen **yhteenvetona** havaittiin, että tehtävän suuremman kokoluokan muutoksen vuoksi kielimalleilla ei kyetty luomaan toimivaa testiympäristöä kolmella syötteellä ja lisäselvittelyiden jälkeen. Annetut syötteet riippuivat edellisestä saadusta vastauksesta, minkä vuoksi saadut vastaukset ovat toisista poikkeavia, mutta sisällöltään vastauksien rakenteet olivat samankaltaisia. Yhtenä erona havaittiin GitHub Copilotin vastaukset, jotka olivat muihin malleihin nähden lyhyempinä.

Jokainen kielimallin vastaus testiympäristön luomiseen sisälsivät samoja tekniikoita. Testikehykseksi ehdotettiin Jest ja React Testing Library, jotka ovat suosittuja työkaluja testien luomiseen React-sovelluksille. Jest voidaan asentaa myös muille projekteille, jotka eivät käytä React-kirjastoa. Näitä tukemaan ehdotettiin asennettavaksi muun muassa "vite-jest", "@testing-library/jest-dom", "babel-jest", "@babel/core", "@babel/preset-env" ja "@babel/preset-react". Moni ehdotetuista lisäkirjastoista ovat pääkäytössä olevien kirjastojen tukevia kirjastoja. Hallusinaatiota havaittiin esimerkiksi Copilot:n kohdalla, kun asennettavana kirjastona ehdotettiin tuntematonta "vite-plugin-testing-library", jota ei löydy NPM-paketinhallinnasta.

Kaikkien kolmen kielimallin kohdalla kohdattiin ongelma ESLint-analysointityökalun kanssa. Syötteisiin saadut vastaukset sisälsivät muutamissa tapauksissa ongelmakohtia, jotka ESLint havaitsi nostattaen virheilmoituksia.

## 7. TULOKSET

Kahdella luodulla haasteella testatut, suuria kielimalleja hyödyntävät sovellukset antoivat käyttäjälle ohjaavia neuvoja ja vastauksia annettuihin syötteisiin. Jo kahden erilaajuisen haasteen avulla havaittiin, että pyrittäessä pieneen määrään syötteitä toimivimmat ratkaisut saatiin haasteelle, mikä oli kooltaan pienempi sisältäen selkeämmän rajauksen ja kuvaavamman alkusyötteen. Tyylianimaatiohaasteessa kielimallille rajattiin selkeästi ratkaistava ongelma, kuinka toiminnallisuuden pitää toimia sekä jaettiin React-komponentin koodi ja sen tyyliääritelmät. Testausympäristön luomisessa taas alkusyötteessä määritettiin tyypistetyksi ratkaistava ongelma ja käytettävät tekniikat ilman tarkempia tietoja tekniikoiden ja kirjastojen käytetyistä versioista.

Rakenteeltaan kielimallien antamat vastaukset olivat samankaltaisia. Vastauksien rakenne muuttui hieman syötteiden sisällön perusteella. Kysyttäessä ohjeita toiminnallisuuden toteuttamiseen vastauksien sisältö rajautui kolmeen osaan: tehtävänannon tiivistelmä, ehdotetut tehtävät muutokset ja loppuyhteenvedo ehdotetuista muutoksista. Ehdotetut tehtävät muutokset saattoivat sisältää askelohjeet ominaisuuden tai korjauksen toteuttamiseen. Vaikka jokaisen kielimallin kohdalla havaittiin ongelmia, esimerkiksi hallusinointia, kykenivät ne opastamaan käyttäjää ominaisuuksien toteuttamisessa ja ongelmien ratkaisemisessa.

Kielimalleja hyödyntävien sovelluksien välillä ei kokonaiskuvassa nähty suuria eroavaisuuksia (taulukko 9), vaikkakin ensimmäisessä haasteessa, tyylianimaation luomisessa, tuntemattomasta syystä Google Bard ei kyennyt vastaamaan käyttäjän antamiin syötteisiin. Samassa haasteessa ChatGPT:n ja Copilotin avulla saatiin toteutettua haluttu lopputulos, mikä vaati kolmen kielimallin antaman vastauksen jälkeen käyttäjältä aikaa ja tutkimista ominaisuuksien korjaamiseksi.

Toisessa haasteessa jokainen kielimalli antoi tukea käyttäjälle, jolla ei ollut aiempaa kokemusta testiympäristön perustamisesta. Vastauksissa ehdotettiin asennettavia paketteja, tiedostoja, joita käyttäjän pitäisi luoda tai muokata, ja miten saatuja ja raportoituja virheilmoituksia pitäisi yrittää ratkoa. Tästä huolimatta kolme syötettä, vastausta ja käyttäjän oma panostus ongelmien tutkimiseen ja korjaamiseen ei saanut pelastettua toiminnallisuutta toimintakuntoon. Yhtenä ongelman aiheuttajana oli projektissa virheiden ja ongelmien havaitsemiseen käytössä ollut ESLint-työkalu, jonka ongelmiin kielimallit koittivat löytää ratkaisuja onnistumatta siinä.

Se, kuinka kielimallit onnistuvat, riippuu avainsanojen esiintyvyyden määrästä kielimalleille opetetussa datassa. Uusi tekniikka tai tieto, jota ei olla opetettu kielimallille, tekee kielimallista kykenemättömän vastaamaan kyseiseen tekniikkaan ja tietoon liittyviin kysymyksiin. Mitä enemmän tietoa löytyy opetusdatasta, sitä parempia ja oikeampia vastauksia kielimalli antaa. Lisäksi vastauksien laatuun vaikuttavat kielimallille annettujen syötteiden ja kehotteiden laatu ja ratkaistavan ongelman laajuus.

Kolmesta kielimallista ChatGPT antoi pidempiä vastauksia sisällyttäen niihin enemmän merkkejä verrattuna Google Bard:iin tai GitHub Copilot:iin. Suurin ero näissä oli GitHub Copilotiin verrattuna, sillä ChatGPT:n vastaus toisen haasteen ensimmäiseen syötteeseen oli 363 sanan (2634 merkkiä) pituinen ja Copilot:n vastaus 229 sanaa (1579 merkkiä). Sanojen tai merkkien lukumäärä ei kerro vastauksen laadusta, mutta saattaa kertoa eri kielimallien opettamisesta ja hienosäätämisestä eri tarkoituksiin. Siinä missä ChatGPT ja Google Bard ovat kohdistettu yleisempään käyttöön, GitHub Copilot on koulutettu ja hienosäädetty kehittäjien työkaluksi.

Yhtenä syynä laajempiin vastauksiin voi olla kielimallien muistittomuus, jolloin uusien vastauksien luomisessa uudemman syötteen lisäksi vanhemmat keskustelut sisällytetään syötteeseen. Kaikkea keskustelun tekstejä ei välttämättä kyetä lisäämään syötteeseen, sillä kielimalleilla on tokeniyläraja syötteen käsittelylle. Tämä ongelma esiintyy vanhempien kielimallien kohdalla.

Tekoälyjen muistien saralla on tulossa muutoksia. OpenAI julkaisi helmikuussa 2024 ChatGPT:lle kyvyn muistaa aiempia keskusteluja ja hyödyntää näitä myös uusissa keskusteluissa [110], jos ChatGPT:lle on antanut luvan hyödyntää aikaisempia keskusteluja.

Työn tutkimuskysymyksen näkökulmasta katsottuna, kolmea kielimallia vertaamalla mikään yksittäinen malli ei noussut selkeästi toista paremmin ja siten korkeammalle sijalle muihin nähden. Huomioitavaa on myös, että näistä kolmesta mallista haasteiden toteutusvaiheessa Google Bard oli vielä kehitysvaiheessa, mikä saattoi aiheuttaa toimimattomuutta ensimmäisen haasteen kohdalla. Google Bard ja ChatGPT taas ovat suunniteltuja yleisempää käyttöä varten kyeten myös vastaamaan ohjelmoijia askarruttaviin kysymyksiin. Yleiseltä tasolta katsottuna GitHub Copilot on suunnattu kehittäjille, eikä ole yhtä helposti saatavilla kuin Google Bard tai OpenAI ChatGPT.

Kun sovelluksia verrataan muiden näkökulmien mukaan, aletaan huomaamaan eroavaisuuksia. Sovellukset tarjoavat erilaisia tilaussuunnitelmia, jotka sisältävät eri ominaisuuksia. Valintaa pohtiessaan käyttäjää suositellaan tutustumaan eri sovelluksien ominaisuuksiin ja tilaussuunnitelmien hintoihin, ja pohtimaan sitä kautta, mikä sovellus sopii

hänelle parhaiten. Palveluiden hintatason perusteella katsottuna Google Bard ja ChatGPT saattavat houkutella enemmän käyttäjiä ilmaisuudellaan. Maksullisissa tilaussuunnitelmissa huomioitavana ovat Googlen tarjoamat muut tuotteet kuten suurempi Google Drive -tallennustila. Myös OpenAI tarjoaa muita työkaluja ChatGPT:n lisäksi. Lisäksi maksulliset suunnitelmat antavat käyttäjille mahdollisuuden hyödyntää kyvykkäämpiä kielimalleja.

Jokainen sovelluksista soveltuu monipuoliseen käyttöön; Bard ja ChatGPT kykenevät myös antamaan vastauksia ohjelmointiin liittyviin kysymyksiin. Jos käyttäjä haluaa hyödyntää työkalua pelkästään sovelluskehityksessä, silloin Copilot tarjoaa paremmat ominaisuudet. Tiimien ja organisaatioiden näkökulmasta katsottuna ChatGPT ja Copilot tarjoavat sopimussuunnitelmia, jotka soveltuvat parhaiten tiimeille ja organisaatioille.

Omistajuuden ja tietoturvan näkökulmasta katsottuna OpenAI kertoo käyttäjän omistavan syötteet sekä kielimallien antamat tulokset. GitHub kertoo, etteivät tule vaatimaan itselleen annettujen ehdotuksien omistajuutta, mutta painottavat, että useammalla käyttäjällä on mahdollisuus luoda vastaavanlaisia tuloksia keskenään. Googlen dokumenteista ei käy ilmi se, kuka omistaa syötteet ja generoidut vastaukset, mutta tiedoissa kerrotaan, ettei Googlen palveluun kannata syöttää mitään luottamuksellista tai arkaluonteista tietoa. Taulukko 9 esittelee yhteenvedon tutkimuksessa tarkasteltujen kielimalleja hyödyntävien sovellusten vertailusta.

Taulukko 9. Yhteenveto kielimalleja hyödyntävistä sovelluksista

|  | Google Bard  | OpenAI ChatGPT   | GitHub Copilot  |
|--|--|--|---|
| Kielimalli   | LaMDA 2  | GPT-3.5  | OpenAI Codex (GPT-3)  |
| Haasteista suoriutuminen:<br>1) Haaste 1: Tyyliani-<br>maation luominen<br>2) Haaste 2: Testi-<br>ympäristön ja testien luominen | 1) Ei kyennyt vastaamaan käyttäjän antamaan syötteisiin.<br>2) Tuki käyttäjää testi-<br>ympäristön ja testien luomisessa. Lop-<br>putuloksena sovellus eikä testit<br>toimineet. | 1) Haluttu lopputulos saatiin<br>toteutettua pienillä muutok-<br>silla.<br>2) Tuki käyttäjää testi-<br>ympäristön ja testien luomisessa.<br>Lopputuloksena sovellus<br>eikä testit toimineet.    | 1) Haluttu lopputulos saatiin<br>toteutettua pienillä muutok-<br>silla.<br>2) Tuki käyttäjää testi-<br>ympäristön ja testien luomisessa.<br>Lopputuloksena sovellus<br>eikä testit toimineet. |
| Tilaussuunnitelmat   | Ilmainen ja maksullinen. Lisä-<br>tilaa Google Driveen.  | Ilmainen, kolme erilaista<br>kuukausi- tai vuosimaksul-<br>lista. Tiimeille ja organisaa-<br>tioille suunnattuja tilaus-<br>suunnitelmaa. Maksullinen<br>tuo muita OpenAI tuotteita<br>käyttöön. | Ilmainen opiskelijoille. Mak-<br>sullinen kuukausi- tai vuo-<br>simaksullinen.  |
| Tietoturvan huomioita  | Ei suosittele syöttämään arka-<br>luonteista ja henkilökohtaisia<br>tietoja.   |  |   |
| Omistajuuden huomi-<br>oita  | Ei kerrota dokumenteissa   | Käyttäjä omistaa syötteet<br>sekä luodut tulosteet, va-<br>rauksin.  | GitHub ei vaadi itselleen<br>luotujen tulosteiden omista-<br>juutta, varauksin.   |
| Datan keräämisen<br>huomioita  | Palvelun kehittämiseen. Data<br>anonymisoitu.  | Palvelun kehittämiseen.<br>Käyttäjä voi kieltää datan<br>käytön kielimallin opettami-<br>sessa. Team- ja Enterprise<br>dataa ei käytetä opettami-<br>seen.                                       | Palvelun kehittämiseen.<br>Data pseudonymisoitu.  |
| Soveltuvuus arkielä-<br>män kysymyksiin  | Monipuolinen kielimalli, verkko-<br>sovellus, helposti saavutetta-<br>vissa.   | Monipuolinen kielimalli,<br>verkkosovellus, helposti<br>saavutettavissa.   | Kohdistettu kehittäjille.<br>Vaatii asentamisen teks-<br>tieditorille.  |
| Soveltuvuus työelä-<br>män kysymyksiin   | Tukee sovelluskehittäjää, mutta<br>datan keräämisessä kysymyk-<br>siä.   | Tukee sovelluskehittäjää.<br>Mahdollisuus kieltää tiedon<br>keräämisen ja sillä opetta-<br>misen.  | Tukee sovelluskehittäjää.<br>Tarjoaa monia ominaisuuksia<br>sovelluskehittäjille.   |

Suurin painoarvo palvelua valittaessa on sen tarjoamien ominaisuuksien hyödyllisyys juuri siihen tarkoitukseen, mihin kielimallia tarjoavaa palvelua etsitään. GitHub Copilot on nopeasti saatavilla tekstieditorissa tarjoten myös muita hyödyllisiä työkaluja, joita tässä työssä ei nostettu suuremmin esille. Keskustelunäkymän lisäksi Copilot tarjoaa muun muassa ennakoivan tekstintäydentämisen kooditiedostoissa, kommenttien luomisen valitulle tekstille ja ajettavan koodin generoimisen perustuen kirjoitettuun kommenttiin.

Hyödyntäessä tekoälyjä ja kielimalleja käyttäjän kannattaa muistaa, että kielimallit tuottavat vastauksia lähdedatan perusteella. On mahdollista, että lähdedataan on sisällytetty tekijäoikeudellista tai lisensoitua informaatiota vahingossa. Käyttäjällä on lopullinen vastuu analysoida saatu vastaus, jotta vältytään tilanteelta, jossa vahingossa käytetään tekijäoikeudellisesti suojattua tai lisensoitua tietoa.

## 8. POHDINTA

Diplomityön tavoitteena oli tutkia helposti saatavilla olevia, kielimalleja hyödyntäviä keskustelusovelluksia sovelluskehittäjän työkaluna. Työssä käytiin läpi tekoälyjen ja kielimallien historiaa, kielimallien nopeaa kehittymistä sekä tutustuttiin Google Bard -, GitHub Copilot - sekä OpenAI ChatGPT -sovelluksiin. Sovellukset valittiin rajaamalla sovellukset helposti saataviin verkkopalveluihin. Tekoälyjä sekä kielimalleja voidaan suorittaa paikallisesti laitteella, mutta niiden koon ja tehoa vaativan suorittamisen vuoksi työssä rajauduttiin verkkopalveluihin.

Aiheen valitseminen diplomityölle oli haasteellista. Lopulta valintaa helpotti yleinen kiinnostuminen luovaan tekoälyyn, josta uutisoitiin runsaasti vuosien 2022 ja 2023 aikana. Vaikka lähtökohtaisesti kiinnostus kohdistui tekstistä-kuvaksi-kielimalleihin, alkupe räiseksi näkökulmaksi valikoitui kielimallin hyödyntäminen osana sovelluskehittäjän työtä, jota kirjoittaja tekee työkseen. Diplomityön edetessä tutkimuksen kohde tarkentui jatkuvaan kehittämiseen ja pienkehitykseen. Etenkin kysymys tekoälyjen hyödyntämisestä yhtenä sovelluskehittäjien työkaluna moniprojektisessa työympäristössä kiinnosti. Kehittäjä, joka IT-talossa on osa ylläpito- tai jatkuvan kehittämisen tiimiä, saattaa olla monessa pienkehitysprojekteissa mukana samaan aikaan. Monessa projektissa toimiminen samanaikaisesti vaatii projektien priorisoimista, ajan hallintaa, nopeaa tiedon omaksumista sekä mahdollisesti vähäisten tuntimäärien vuoksi nopeaa tuloksen luomista, mikä voi tehdä työstä kuormittavaa. Tulevaisuudessa tekoäly voi tarjota erilaisia ratkaisuja kehittäjän haasteelliseen työhön.

Vertailevaan tutkimukseen luodut kaksi haastetta olivat kooltaan ja vaikeusasteiltaan erilaiset. Ajan loppumisen vuoksi työtä varten ei ehditty luomaan kolmatta haastetta, jonka avulla sovelluksien vertailu olisi ollut helpompaa. Lisäksi kolmen sovelluksen lisäksi olisi ollut kiinnostavaa tutkia muitakin kielimalleja sekä niitä hyödyntäviä sovelluksia ja palveluita. Yhtenä kielimallina olisi ollut kiinnostavaa tutkia Google Geminiä ja verrata tätä aiempaan Google Bard - ja sen käyttämään LaMDA 2 -kielimalliin. Myös näitä malleja olisi voinut tutkia kehittäjän näkökulmasta arvioiden, tuoko Gemini-kielimalliperhe muutoksia LaMBDA 2:een verrattuna.

Työn laajuuden pitämiseksi maltillisena työlle piti määrittää rajat, joiden avulla valittiin testattavat sovellukset. Kielimallien arvioinnin haasteena oli myös alan nopea kehittyminen. Tutkimuksen aikana työkalut saivat monia päivityksiä muun muassa kielimallin ja käyttöliittymän osalta.

Kielimallit ovat verrattaen uutta ja nopeasti kehittyvä tutkimusala. Tulevaisuudessa yhtenä kiinnostavana näkökulmana olisi tutkia pienkehittämisen kuormittavuutta ohjelmistoalalla ja kielimallien hyödyntämien mahdollisuuksia erityisesti työkuorman keventämisen tarpeesta käsin. Myös syvempi katsaus kielimallien, tekoälyn ja niitä hyödyntävien sovelluksien ja palveluiden datan käytöstä herättää kysymyksiä tutkittavaksi: missä määrin kielimallien käyttämä data on tekijänoikeusvapaata ja minkälainen on riski sille, että käyttäjä luo tekoälyä käyttäessään jotain sellaista, mikä on saatu opettamalla kielimallia lisensoidulla tai tekijänoikeudellisella tiedolla.

Kuka tällöin oikeasti omistaa koodin?

# LÄHTEET

- [1] Marr B. A Short History Of ChatGPT: How We Got To Where We Are Today [Internet]. Forbes. [viitattu 7. kesäkuuta 2023]. Saatavissa: <https://www.forbes.com/sites/bernardmarr/2023/05/19/a-short-history-of-chatgpt-how-we-got-to-where-we-are-today/>
- [2] Google vastaa Microsoftin haasteeseen tekoälyssä, julkaisee oman kilpailijansa ChatGPT:lle [Internet]. Yle Uutiset. 2023 [viitattu 8. kesäkuuta 2023]. Saatavissa: <https://yle.fi/a/74-20016690>
- [3] Noy S, Zhang W. Experimental evidence on the productivity effects of generative artificial intelligence. Science. 14. heinäkuuta 2023;381(6654):187–92.
- [4] A Complete History of Artificial Intelligence [Internet]. G2. [viitattu 24. heinäkuuta 2023]. Saatavissa: <https://www.g2.com/articles/history-of-artificial-intelligence>
- [5] Siukonen T, Neittaanmäki P. Mitä tulisi tietää tekoälystä. Jyväskylä: Docendo Oy; 2019.
- [6] What is Artificial Intelligence (AI) ? | IBM [Internet]. [viitattu 24. heinäkuuta 2023]. Saatavissa: <https://www.ibm.com/topics/artificial-intelligence>
- [7] Artificial intelligence - Alan Turing, AI Beginnings | Britannica [Internet]. [viitattu 24. heinäkuuta 2023]. Saatavissa: <https://www.britannica.com/technology/artificial-intelligence/Alan-Turing-and-the-beginning-of-AI>
- [8] Turing test | Definition & Facts | Britannica [Internet]. 2023 [viitattu 24. heinäkuuta 2023]. Saatavissa: <https://www.britannica.com/technology/Turing-test>
- [9] SITNFlash. The History of Artificial Intelligence [Internet]. Science in the News. 2017 [viitattu 24. heinäkuuta 2023]. Saatavissa: <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>
- [10] Newell A, Simon H. The logic theory machine—A complex information processing system. IRE Trans Inf Theory. syyskuuta 1956;2(3):61–79.
- [11] heuristinen - Kielitoimiston sanakirja [Internet]. [viitattu 31. heinäkuuta 2023]. Saatavissa: <https://www.kielitoimistonsanakirja.fi/#/heuristinen%20?searchMode=all>
- [12] McCarthy J. WHAT IS ARTIFICIAL INTELLIGENCE? 12. marraskuuta 2007;
- [13] Russell Stuart, Norvig Peter. Artificial Intelligence. 3rd ed. NOIDA: Pearson Education Limited; 2016.
- [14] Sharma L, Garg PK. Artificial intelligence : technologies, applications, and challenges. First edition. Boca Raton: CRC Press, Taylor & Francis Group; 2022.
- [15] What is Strong AI? | IBM [Internet]. [viitattu 21. kesäkuuta 2023]. Saatavissa: <https://www.ibm.com/topics/strong-ai>

- [16] Miten musta aukko syntyy? Entä miten sing - Tietysti.fi [Internet]. [viitattu 20. helmikuuta 2024]. Saatavissa: <https://www.aka.fi/tietysti/kysy-tieteesta/miten-musta-aukko-syntyy-enta-miten-singulariteetti-syntyy/>
- [17] The Singularity [Internet]. CraigBellamy.net(.au). 2023 [viitattu 20. helmikuuta 2024]. Saatavissa: <https://www.craigbellamy.net/2023/03/03/the-singularity/>
- [18] Artificial intelligence (AI) | Definition, Examples, Types, Applications, Companies, & Facts | Britannica [Internet]. 2023 [viitattu 20. kesäkuuta 2023]. Saatavissa: <https://www.britannica.com/technology/artificial-intelligence>
- [19] What Is an Expert System? | Definition from TechTarget [Internet]. Enterprise AI. [viitattu 12. kesäkuuta 2023]. Saatavissa: <https://www.techtarget.com/searchenterpriseai/definition/expert-system>
- [20] OpenAI. Improving language understanding with unsupervised learning [Internet]. [viitattu 11. heinäkuuta 2023]. Saatavissa: <https://openai.com/research/language-unsupervised>
- [21] Koneoppiminen | Ite wikin digitalisoinnin opas [Internet]. 2018 [viitattu 12. kesäkuuta 2023]. Saatavissa: <https://www.itewiki.fi/opus/koneoppiminen/>
- [22] What is Deep Learning? | IBM [Internet]. [viitattu 13. kesäkuuta 2023]. Saatavissa: <https://www.ibm.com/topics/deep-learning>
- [23] Generative AI: How It Works, History, and Pros and Cons [Internet]. Investopedia. [viitattu 10. heinäkuuta 2023]. Saatavissa: <https://www.investopedia.com/generative-ai-7497939>
- [24] Murphy KP. Probabilistic Machine Learning: An introduction [Internet]. MIT Press; 2022. Saatavissa: [probml.ai](http://probml.ai)
- [25] Krogh A. What are artificial neural networks? Nat Biotechnol. helmikuuta 2008;26(2):195–7.
- [26] Heli Tuominen. Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM [Internet]. [viitattu 7. toukokuuta 2024]. Saatavissa: <https://tim.jyu.fi/view/143092>
- [27] What are Neural Networks? | IBM [Internet]. [viitattu 25. heinäkuuta 2023]. Saatavissa: <https://www.ibm.com/topics/neural-networks>
- [28] Li Z, Wang C, Han M, Xue Y, Wei W, Li LJ, ym. Thoracic Disease Identification and Localization with Limited Supervision [Internet]. arXiv; 2018 [viitattu 20. helmikuuta 2024]. Saatavissa: <http://arxiv.org/abs/1711.06373>
- [29] Liu Y, Gadepalli K, Norouzi M, Dahl GE, Kohlberger T, Boyko A, ym. Detecting Cancer Metastases on Gigapixel Pathology Images [Internet]. arXiv; 2017 [viitattu 20. helmikuuta 2024]. Saatavissa: <http://arxiv.org/abs/1703.02442>
- [30] Speech and Language Processing [Internet]. [viitattu 16. kesäkuuta 2023]. Saatavissa: <https://web.stanford.edu/~jurafsky/slp3/>
- [31] Aydın Ö, Karaarslan E. Is ChatGPT Leading Generative AI? What is Beyond Expectations? [Internet]. Rochester, NY; 2023 [viitattu 27. kesäkuuta 2023]. Saatavissa: <https://papers.ssrn.com/abstract=4341500>

- [32] What is Natural Language Processing? | IBM [Internet]. [viitattu 27. kesäkuuta 2023]. Saatavissa: <https://www.ibm.com/topics/natural-language-processing>
- [33] Elazar Y, Kassner N, Ravfogel S, Ravichander A, Hovy E, Schütze H, ym. Measuring and Improving Consistency in Pretrained Language Models. *Trans Assoc Comput Linguist.* 6. joulukuuta 2021;9:1012–31.
- [34] Language Models Pre-training [Internet]. [viitattu 30. kesäkuuta 2023]. Saatavissa: <https://kaggle.com/code/vad13irt/language-models-pre-training>
- [35] Zhao WX, Zhou K, Li J, Tang T, Wang X, Hou Y, ym. A Survey of Large Language Models [Internet]. arXiv; 2023 [viitattu 3. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/2303.18223>
- [36] Salian I. NVIDIA Blog: Supervised Vs. Unsupervised Learning [Internet]. NVIDIA Blog. 2018 [viitattu 26. helmikuuta 2024]. Saatavissa: <https://blogs.nvidia.com/blog/supervised-unsupervised-learning/>
- [37] What is Supervised Learning? | IBM [Internet]. [viitattu 26. helmikuuta 2024]. Saatavissa: <https://www.ibm.com/topics/supervised-learning>
- [38] What are Large Language Models? | NVIDIA Glossary [Internet]. NVIDIA. [viitattu 26. helmikuuta 2024]. Saatavissa: <https://www.nvidia.com/en-us/glossary/large-language-models/>
- [39] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright CL, Mishkin P, ym. Training language models to follow instructions with human feedback [Internet]. arXiv; 2022 [viitattu 17. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/2203.02155>
- [40] What is ChatGPT? | OpenAI Help Center [Internet]. [viitattu 17. heinäkuuta 2023]. Saatavissa: <https://help.openai.com/en/articles/6783457-what-is-chatgpt>
- [41] Salvagno M, Taccone FS, Gerli AG. Artificial intelligence hallucinations. *Crit Care.* 10. toukokuuta 2023;27(1):180.
- [42] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, ym. Attention Is All You Need [Internet]. arXiv; 2017 [viitattu 10. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/1706.03762>
- [43] Radford A, Narasimhan K, Salimans T, Sutskever I. Improving Language Understanding by Generative Pre-Training.
- [44] Menon P. Introduction to Large Language Models and the Transformer Architecture [Internet]. Medium. 2023 [viitattu 4. heinäkuuta 2023]. Saatavissa: <https://rpradeepmenon.medium.com/introduction-to-large-language-models-and-the-transformer-architecture-534408ed7e61>
- [45] Saeed M. A Gentle Introduction to Positional Encoding in Transformer Models, Part 1 [Internet]. MachineLearningMastery.com. 2022 [viitattu 6. heinäkuuta 2023]. Saatavissa: <https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/>
- [46] PhD SR. Understanding Encoder And Decoder LLMs [Internet]. [viitattu 10. heinäkuuta 2023]. Saatavissa: <https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder>

- [47] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, ym. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [Internet]. arXiv; 2020 [viitattu 11. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/1910.10683>
- [48] How to Use Softmax Function for Multiclass Classification [Internet]. [viitattu 22. helmikuuta 2024]. Saatavissa: <https://www.turing.com/kb/softmax-multiclass-neural-networks>
- [49] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016.
- [50] Encoder models - Hugging Face NLP Course [Internet]. [viitattu 10. heinäkuuta 2023]. Saatavissa: <https://huggingface.co/learn/nlp-course/chapter1/5>
- [51] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Internet]. arXiv; 2019 [viitattu 10. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/1810.04805>
- [52] Yang J, Jin H, Tang R, Han X, Feng Q, Jiang H, ym. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond [Internet]. arXiv; 2023 [viitattu 3. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/2304.13712>
- [53] OpenAI. GPT models [Internet]. [viitattu 14. heinäkuuta 2023]. Saatavissa: <https://platform.openai.com/docs/guides/gpt>
- [54] Introducing OpenAI [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://openai.com/blog/introducing-openai>
- [55] About [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://openai.com/about>
- [56] Zhu Y, Kiros R, Zemel R, Salakhutdinov R, Urtasun R, Torralba A, ym. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books [Internet]. arXiv; 2015 [viitattu 12. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/1506.06724>
- [57] Chelba C, Mikolov T, Schuster M, Ge Q, Brants T, Koehn P, ym. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling [Internet]. arXiv; 2014 [viitattu 12. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/1312.3005>
- [58] Solaiman I, Brundage M, Clark J, Askeel A, Herbert-Voss A, Wu J, ym. Release Strategies and the Social Impacts of Language Models [Internet]. arXiv; 2019 [viitattu 13. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/1908.09203>
- [59] Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, ym. Language Models are Few-Shot Learners [Internet]. arXiv; 2020 [viitattu 13. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/2005.14165>
- [60] Child R, Gray S, Radford A, Sutskever I. Generating Long Sequences with Sparse Transformers [Internet]. arXiv; 2019 [viitattu 13. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/1904.10509>
- [61] OpenAI. Models [Internet]. [viitattu 13. heinäkuuta 2023]. Saatavissa: <https://platform.openai.com/docs/models/overview>
- [62] Introducing ChatGPT [Internet]. [viitattu 14. heinäkuuta 2023]. Saatavissa: <https://openai.com/blog/chatgpt>

- [63] OpenAI. GPT-4 Technical Report [Internet]. arXiv; 2023 [viitattu 14. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/2303.08774>
- [64] ChatGPT — Release Notes | OpenAI Help Center [Internet]. [viitattu 18. heinäkuuta 2023]. Saatavissa: <https://help.openai.com/en/articles/6825453-chatgpt-release-notes>
- [65] GPT-4 API general availability and deprecation of older models in the Completions API [Internet]. [viitattu 27. helmikuuta 2024]. Saatavissa: <https://openai.com/blog/gpt-4-api-general-availability>
- [66] OpenAI Platform [Internet]. [viitattu 27. helmikuuta 2024]. Saatavissa: <https://platform.openai.com>
- [67] Aligning language models to follow instructions [Internet]. [viitattu 16. kesäkuuta 2023]. Saatavissa: <https://openai.com/research/instruction-following>
- [68] Google I/O 2022: Advancing knowledge and computing [Internet]. Google. 2022 [viitattu 18. heinäkuuta 2023]. Saatavissa: <https://blog.google/technology/developers/io-2022-keynote/>
- [69] Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance [Internet]. 2022 [viitattu 28. helmikuuta 2024]. Saatavissa: <https://blog.research.google/2022/04/pathways-language-model-palm-scaling-to.html?m=1>
- [70] Introducing PaLM 2 [Internet]. Google. 2023 [viitattu 5. maaliskuuta 2024]. Saatavissa: <https://blog.google/technology/ai/google-palm-2-ai-large-language-model/>
- [71] LaMDA: our breakthrough conversation technology [Internet]. Google. 2021 [viitattu 18. heinäkuuta 2023]. Saatavissa: <https://blog.google/technology/ai/lamda/>
- [72] Thoppilan R, De Freitas D, Hall J, Shazeer N, Kulshreshtha A, Cheng HT, ym. LaMDA: Language Models for Dialog Applications [Internet]. arXiv; 2022 [viitattu 18. heinäkuuta 2023]. Saatavissa: <http://arxiv.org/abs/2201.08239>
- [73] Towards a Conversational Agent that Can Chat About...Anything [Internet]. 2020 [viitattu 18. heinäkuuta 2023]. Saatavissa: <https://ai.googleblog.com/2020/01/towards-conversational-agent-that-can.html>
- [74] An important next step on our AI journey [Internet]. Google. 2023 [viitattu 18. heinäkuuta 2023]. Saatavissa: <https://blog.google/technology/ai/bard-google-ai-search-updates/>
- [75] Try Bard and share your feedback [Internet]. Google. 2023 [viitattu 28. helmikuuta 2024]. Saatavissa: <https://blog.google/technology/ai/try-bard/>
- [76] Introducing Gemini: our largest and most capable AI model [Internet]. Google. 2023 [viitattu 28. helmikuuta 2024]. Saatavissa: <https://blog.google/technology/ai/google-gemini-ai/>
- [77] Bard becomes Gemini: Try Ultra 1.0 and a new mobile app today [Internet]. Google. 2024 [viitattu 28. helmikuuta 2024]. Saatavissa: <https://blog.google/products/gemini/bard-gemini-advanced-app/>

- [78] About GitHub Copilot Individual [Internet]. GitHub Docs. [viitattu 5. maaliskuuta 2024]. Saatavissa: <https://docs.github.com/en/copilot/copilot-individual/about-github-copilot-individual>
- [79] Friedman N. Introducing GitHub Copilot: your AI pair programmer [Internet]. The GitHub Blog. 2021 [viitattu 24. huhtikuuta 2024]. Saatavissa: <https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/>
- [80] OpenAI Codex [Internet]. [viitattu 8. huhtikuuta 2024]. Saatavissa: <https://openai.com/blog/openai-codex>
- [81] ChatGPT [Internet]. [viitattu 18. heinäkuuta 2023]. Saatavissa: <https://chat.openai.com>
- [82] Press [Internet]. GitHub. [viitattu 5. maaliskuuta 2024]. Saatavissa: <https://github.com/about/press>
- [83] GitHub features: the right tools for the job [Internet]. GitHub. [viitattu 5. maaliskuuta 2024]. Saatavissa: <https://github.com/features>
- [84] GitHub Copilot – November 30th Update [Internet]. The GitHub Blog. 2023 [viitattu 24. huhtikuuta 2024]. Saatavissa: <https://github.blog/changelog/2023-11-30-github-copilot-november-30th-update/>
- [85] Beuke F. GitHub 2.0: GitHub Language Statistics [Internet]. 2023. Saatavissa: <https://madnight.github.io/github/#/>
- [86] GitHub Copilot overview [Internet]. [viitattu 5. toukokuuta 2024]. Saatavissa: <https://code.visualstudio.com/docs/copilot/overview>
- [87] OpenAI [@OpenAI]. ChatGPT can now browse the internet to provide you with current and authoritative information, complete with direct links to sources. It is no longer limited to data before September 2021. <https://t.co/pyj8a9HWkB> [Internet]. Twitter. 2023 [viitattu 8. toukokuuta 2024]. Saatavissa: <https://twitter.com/OpenAI/status/1707077710047216095>
- [88] Pricing [Internet]. [viitattu 15. huhtikuuta 2024]. Saatavissa: <https://openai.com/pricing>
- [89] ChatGPT Pricing [Internet]. [viitattu 15. huhtikuuta 2024]. Saatavissa: <https://openai.com/chatgpt/pricing>
- [90] Introducing GPTs [Internet]. [viitattu 15. huhtikuuta 2024]. Saatavissa: <https://openai.com/blog/introducing-gpts>
- [91] Sign in to the Gemini web app - Gemini Apps Help [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://support.google.com/gemini/answer/13278668?hl=en>
- [92] Mitä Gemini Apps voi tehdä ja muuta usein kysyttyä [Internet]. Gemini. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://gemini.google.com/faq>
- [93] Google One -laskeutumissivu [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: [https://one.google.com/explore-plan/gemini-advanced?utm\\_source=gemini&utm\\_medium=web&utm\\_campaign=exp\\_update&g1\\_landing\\_page=65](https://one.google.com/explore-plan/gemini-advanced?utm_source=gemini&utm_medium=web&utm_campaign=exp_update&g1_landing_page=65)

- [94] GitHub Copilot · Your AI pair programmer [Internet]. GitHub. [viitattu 5. maaliskuuta 2024]. Saatavissa: <https://github.com/features/copilot>
- [95] About billing for GitHub Copilot [Internet]. GitHub Docs. [viitattu 11. huhtikuuta 2024]. Saatavissa: <https://docs.github.com/en/billing/managing-billing-for-github-copilot/about-billing-for-github-copilot>
- [96] Terms of use [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://openai.com/policies/terms-of-use>
- [97] How your data is used to improve model performance | OpenAI Help Center [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://help.openai.com/en/articles/5722486-how-your-data-is-used-to-improve-model-performance>
- [98] What is ChatGPT Team? | OpenAI Help Center [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://help.openai.com/en/articles/8792828-what-is-chatgpt-team>
- [99] What is ChatGPT Enterprise? | OpenAI Help Center [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://help.openai.com/en/articles/8265053-what-is-chatgpt-enterprise>
- [100] Enterprise privacy [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://openai.com/enterprise-privacy>
- [101] OpenAI | Security Portal [Internet]. OpenAI. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://trust.openai.com/>
- [102] CCPA vs. GDPR: Similarities and Differences Explained [Internet]. [viitattu 16. huhtikuuta 2024]. Saatavissa: <https://www.okta.com/blog/2021/04/ccpa-vs-gdpr/>
- [103] SOC 2® - SOC for Service Organizations: Trust Services Criteria [Internet]. [viitattu 15. huhtikuuta 2024]. Saatavissa: <https://www.aicpa-cima.com/topic/audit-assurance/audit-and-assurance-greater-than-soc-2>
- [104] Gemini Apps Privacy Hub - Gemini Apps Help [Internet]. [viitattu 17. huhtikuuta 2024]. Saatavissa: <https://support.google.com/gemini/answer/13594961?hl=en>
- [105] Henkilötietojen pseudonymisointi ja anonymisointi [Internet]. Tietosuojavaltuutetun toimisto. [viitattu 8. toukokuuta 2024]. Saatavissa: <https://tietosuoja.fi/pseudonymisointi-anonymisointi>
- [106] GitHub Copilot Trust Center [Internet]. GitHub Resources. [viitattu 8. toukokuuta 2024]. Saatavissa: <https://resources.github.com/copilot-trust-center/>
- [107] Peng S, Kalliamvakou E, Cihon P, Demirer M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot [Internet]. arXiv; 2023 [viitattu 9. toukokuuta 2024]. Saatavissa: <http://arxiv.org/abs/2302.06590>
- [108] Bard's latest updates: Access Gemini Pro globally and generate images [Internet]. Google. 2024 [viitattu 4. toukokuuta 2024]. Saatavissa: <https://blog.google/products/gemini/google-bard-gemini-pro-image-generation/>
- [109] López Espejel J, Ettifouri EH, Yahaya Alassan MS, Chouham EM, Dahhane W. GPT-3.5, GPT-4, or BARD? Evaluating LLMs reasoning ability in zero-shot setting

and performance boosting through prompts. Nat Lang Process J. 1. joulukuuta 2023;5:100032.

[110] Memory and new controls for ChatGPT [Internet]. [viitattu 4. toukokuuta 2024]. Saatavissa: <https://openai.com/index/memory-and-new-controls-for-chatgpt>

## LIITE A: TYYLIIANIMAATION LUOMISEN ALOITUSSYÖTE (HAASTE 1)

I am trying to create an hover animation for my element. I am using React.

Requirements:

I have an class element called "card". When I hover the element, it should make background image 10% bigger.

Animation should not affect to the size of the "card" div. Animation duration should be 0.5s.

When mouse is moved away, the background image size should change back to normal.

I want to change CSS file only.

Code:

```

`
return (
  <div className="cards">
    {cards?.map((card) => {
      return (
        <div
          className="card"
          key={card.fields.title}
          style={{
            background:
`url(${card.fields.image?.fields.file.url}) no-repeat center`,
          }}
        >
          {card.fields.title && card.fields.description && (
            <div className="card-inner">
              <div className="card-title">
                {card.fields.showTitle && (
                  <h1>{card.fields.title}</h1>
                )}
              </div>
              <div className="card-text">
                <p>{card.fields.description}</p>
              </div>
            </div>
          )}
        </div>
      )}
    )}
  </div>
);
`

```

Style:

```

`
.card {
  @extend %roundedBox;
  display: flex;
  background-size: cover !important;
  padding: 50px 0 0;
  width: 400px;
}
`

```

```
min-height: 500px;

&-inner {
  align-self: flex-end;
  backdrop-filter: blur(8px);
  padding: 0 50px;
}

&-title {
  text-transform: uppercase;
}

&-text,
&-title {
  color: white;
}
}
...
```

## LIITE B: CHATGPT-KESKUSTELU TYYLIIANIMAATION LUOMISEEN

### Vastaus 1:

To achieve the hover animation for your "card" element in React with the specified requirements, you can add a CSS class for the hover effect and use the `:hover` pseudo-class. Additionally, you can utilize the `transition` property to control the animation duration. Here's the modified CSS code:

```

```css
.card {
  @extend %roundedBox;
  display: flex;
  background-size: cover !important;
  padding: 50px 0 0;
  width: 400px;
  min-height: 500px;
  transition: background-size 0.5s ease-in-out; /* Added transition property
*/

  &-inner {
    align-self: flex-end;
    backdrop-filter: blur(8px);
    padding: 0 50px;
  }

  &-title {
    text-transform: uppercase;
  }

  &-text,
  &-title {
    color: white;
  }

  &:hover {
    background-size: 110% !important; /* Increase background size on hover
*/
  }
}
```

```

In this code, I added the `transition` property to control the animation duration for the `background-size` property. The `ease-in-out` timing function provides a smooth transition effect. The `:hover` selector is used to define the styles that will be applied when the mouse hovers over the "card" element.

Make sure to adjust the `background-size` and other values according to your design preferences.

---

### Syöte 2:

Animation is working when I hover the "card" element, but there is no duration. Also, image does not get bigger when mouse is on the element: element

size is the same, but image will appear without cropping, making it to look smaller.

---

### Vastaus 2:

I apologize for the oversight. It seems I missed a crucial detail in the implementation. To achieve the desired effect where the background image gets 10% bigger without affecting the size of the "card" div, you should adjust the `background-size` property using percentages. Here's the corrected CSS:

```

```css
.card {
  @extend %roundedBox;
  display: flex;
  background-size: 100% 100% !important; /* Specify initial background size
*/
  padding: 50px 0 0;
  width: 400px;
  min-height: 500px;
  transition: background-size 0.5s ease-in-out; /* Added transition property
*/

  &-inner {
    align-self: flex-end;
    backdrop-filter: blur(8px);
    padding: 0 50px;
  }

  &-title {
    text-transform: uppercase;
  }

  &-text,
  &-title {
    color: white;
  }

  &:hover {
    background-size: 110% 110% !important; /* Increase both width and height
on hover */
  }
}
```

```

Here, I modified the `background-size` property in both the default and `:hover` states to specify both the width and height percentages. The `background-size: 100% 100%` ensures that the image covers the entire "card" div, and on hover, it increases by 10% both horizontally and vertically.

Please give this a try, and it should produce the intended hover effect.

---

### Syöte 3:

The animation works now and it has correct duration. However, the image's aspect ratio is not correct: image looks more narrow and tall. What might cause this?

---

### Vastaus 3:

If the image's aspect ratio appears incorrect during the hover animation, it's possible that the aspect ratio of the image is not being maintained properly. To ensure the correct aspect ratio, you can set one of the dimensions (either width or height) explicitly and let the other dimension adjust accordingly.

Here's an updated version of the CSS:

```

` ``css
.card {
  @extend %roundedBox;
  display: flex;
  background-size: cover !important;
  padding: 50px 0 0;
  width: 400px;
  min-height: 500px;
  transition: background-size 0.5s ease-in-out; /* Added transition property
*/

  &-inner {
    align-self: flex-end;
    backdrop-filter: blur(8px);
    padding: 0 50px;
  }

  &-title {
    text-transform: uppercase;
  }

  &-text,
  &-title {
    color: white;
  }

  &:hover {
    background-size: 110% 110% !important; /* Increase both width and height
on hover */
  }

  & img {
    width: 100%; /* Ensure image takes up 100% width of its container */
    height: auto; /* Allow the height to adjust proportionally */
  }
}
` ``

```

In this updated code, I added a rule `& img` to ensure that the image inside the "card" div takes up 100% of the width, allowing the height to adjust proportionally. This should help maintain the correct aspect ratio during the hover animation. Adjust the width and height percentages as needed to achieve the desired visual effect.

## LIITE C: GITHUB COPILOT -KESKUSTELU TYYLIIANIMAATION LUOMISEEN

### Vastaus 1:

GitHub Copilot: To achieve the hover effect you want, you can use CSS transitions and transforms. Here's how you can modify your CSS:

```
```css
.card {
  /* ...existing styles... */
  background-size: 100% 100% !important;
  transition: background-size 0.5s ease-in-out;
}

.card:hover {
  background-size: 110% 110% !important;
}
```
```

This will make the background image of the `.card` element 10% bigger when hovered over, and the transition will last 0.5 seconds. The ease-in-out` value makes the transition speed gradually increase at the start and decrease at the end, giving a smooth effect.`

Remember to ensure that the image does not repeat itself, or else the hover effect might not work as expected. If the image is repeating, you can add `background-repeat: no-repeat;` to the `.card` class.`

---

### Syöte 2:

The hover transition works, but image looks too narrow. element's aspect ratio is not the same as image's.

---

### Vastaus 2:

GitHub Copilot: If the aspect ratio of the image is not the same as the element, you can use `background-size: cover;` to make sure the image covers the entire element while maintaining its aspect ratio. However, this might cut off some parts of the image if the aspect ratios are significantly different.

Here's how you can modify your CSS:

```
```css
.card {
  /* ...existing styles... */
  background-size: cover !important;
  transition: transform 0.5s ease-in-out;
}

.card:hover {
```

```

    transform: scale(1.1);
  }
  ...

```

In this case, instead of enlarging the background image, we're enlarging the whole card. The `transform: scale(1.1);` will make the card 10% bigger when hovered over. The transition will still last 0.5 seconds, and the `ease-in-out` value will still give a smooth effect.

Remember to ensure that the image does not repeat itself, or else the hover effect might not work as expected. If the image is repeating, you can add `background-repeat: no-repeat;` to the `.card` class.

---

### Syöte 3:

Now the image looks correct, and hover animation works. But the whole element scales up by 1.1, and not only the background image.

---

### Vastaus 3:

GitHub Copilot: To scale only the background image, you can use a pseudo-element to hold the background image. This way, the hover effect will only apply to the background image and not the entire card. Here's how you can modify your CSS:

```

```scss
.card {
  /* ...existing styles... */
  position: relative;
  overflow: hidden;
  background: none;
}

.card::before {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: inherit;
  background-size: cover !important;
  transition: transform 0.5s ease-in-out;
  z-index: -1;
}

.card:hover::before {
  transform: scale(1.1);
}
```

```

In this case, we're creating a pseudo-element `::before` for the `.card` element. This pseudo-element will inherit the background from the `.card`

element, and we apply the hover effect to this pseudo-element. The `overflow: hidden;` on the `.card` element will ensure that the scaled background image does not overflow the card.

## LIITE D: CHATGPT-KESKUSTELU TESTIYMPÄRISTÖN JA TESTIEN LUOMISEEN

### Syöte 1:

I need to create tests for the React project. The project uses Vite as build tool and contentful-cli to fetch data from Contentful. Yarn works as a package manager.

What I need to create tests?

---

### Vastaus 1:

Creating tests for a React project involves setting up a testing environment and writing test cases to ensure the functionality of your components and application. In your case, since your project uses Vite, Contentful, and Yarn, you'll need to configure the testing environment accordingly. Here's a step-by-step guide:

#### ### 1. Choose a Testing Library/Framework:

##### #### React Testing Library:

- This is a popular choice for testing React applications. It encourages testing from a user's perspective.

#### ### 2. Install Testing Dependencies:

```
```bash
# Jest and React Testing Library
yarn add --dev jest @testing-library/react @testing-library/jest-dom

# Optional: Jest configuration for React
yarn add --dev babel-jest @babel/core @babel/preset-env @babel/preset-react

# Optional: Vite Jest Plugin
yarn add --dev vite-jest
```
```

#### ### 3. Configure Jest:

Create a `jest.config.js` file in the root of your project:

```
```javascript
module.exports = {
  preset: 'ts-jest/presets/js-with-babel',
  testEnvironment: 'jsdom',
  transform: {
    '^.+\\.jsx?$': 'babel-jest',
  },
};
```
```

#### ### 4. Create Test Scripts in `package.json`:

```

```json
"scripts": {
  "test": "jest",
  "test:watch": "jest --watch"
}
```

```

### ### 5. Write Your Tests:

Create a `__tests__` folder in the same directory as your components. For example:

```

```
/src
  /components
    Button.js
  /__tests__
    Button.test.js
```

```

Example test file (`Button.test.js`):

```

```javascript
import React from 'react';
import { render, screen } from '@testing-library/react';
import Button from '../components/Button';

test('renders button text', () => {
  render(<Button label="Click me" />);
  const buttonElement = screen.getByText(/click me/i);
  expect(buttonElement).toBeInTheDocument();
});
```

```

### ### 6. Run Your Tests:

```

```bash
yarn test
```

```

### ### 7. Mocking External Dependencies:

For external dependencies like Contentful, you may need to mock API calls or other functions. Jest provides a powerful mocking system. For asynchronous code, you might want to use `jest.mock` or `jest.spyOn` to mock API calls.

### ### 8. Optional: Configure Vite for Jest:

If you encounter issues running Jest with Vite, you can install `vite-jest`:

```

```bash
yarn add --dev vite-jest
```

```

Then, update your `jest.config.js`:

```

```javascript
module.exports = {
  // ... existing config

```

```

moduleFileExtensions: ['js', 'jsx', 'json', 'vue', 'ts', 'tsx'],
moduleNameMapper: {
  '^vue$': 'vue/dist/vue.common.js',
},
transform: {
  '^.+\\.jsx?$': 'babel-jest',
  '^.+\\.tsx?$': 'ts-jest',
},
};

```

This should help you set up a basic testing environment for your React project using Vite, Contentful, and Yarn. Adjust the configuration according to your specific project structure and needs.

---

## Syöte 2:

I got few issues:

The test file:

- Unsafe call of an `any` typed value.eslint@typescript-eslint/no-unsafe-call
- 'test' is not defined.eslintno-undef

jest.config.js file:

```

Parsing error: ESLint was configured to run on `

```

And when I try to run `yarn run` command, I get error:

```

ReferenceError: module is not defined in ES module scope
This file is being treated as an ES module because it has a '.js' file extension and 'C:\(...)\di\package.json' contains "type": "module". To treat it as a CommonJS script, rename it to use the '.cjs' file extension.
    at file:///C:/(...)/di/jest.config.js:1:1

```

## Vastaus 2:

Let's address each of the issues step by step:

### 1. ESLint Issues in Test File:

#### Unsafe call of an `any` typed value:

This issue is related to TypeScript and ESLint complaining about unsafe use of `any`. To resolve this, you can add a type assertion to inform TypeScript that you are intentionally using `any`. For example:

```
````javascript
```

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import Button from '../components/Button';

test('renders button text', () => {
  render(<Button label="Click me" />);
  const buttonElement = screen.getByText(/click me/i) as HTMLElement; // Add
type assertion
  expect(buttonElement).toBeInTheDocument();
});
```
```

#### 'test' is not defined:

Make sure you have properly imported `test` from `@testing-library/react`. Add the following import statement at the beginning of your test file:

```
```javascript
import { test } from '@jest/globals';
```
```

### 2. Jest Configuration Error:

The error you're encountering indicates a parsing error related to ESLint configuration. It seems like Jest is trying to use ESLint on the `jest.config.js` file, and there might be an issue with the ESLint and TypeScript configuration.

You can try adding the following to your `jest.config.js`:

```
```javascript
module.exports = {
  // ... existing config
  parserOptions: {
    tsconfigRootDir: __dirname,
  },
  // ... other configurations
};
```
```

Make sure to replace `` with the actual path if it appears in your error message.

### 3. ES Module Error:

The error "ReferenceError: module is not defined in ES module scope" suggests that Jest is trying to run your configuration file as an ES module. To resolve this, you can either change the file extension of `jest.config.js` to `.cjs` or add `"type": "commonjs"` to your `package.json`.

#### Option 1: Change File Extension

Rename `jest.config.js` to `jest.config.cjs`.

#### Option 2: Update `package.json`

Add the following to your `package.json`:

```
```json
"type": "commonjs",
```
```

After addressing these issues, try running your tests again using `yarn test`. Adjust the solutions based on your project's specific setup and requirements.

---

### Syöte 3:

Yarn test gives two errors:

- Unknown option "parserOptions" with value {"tsconfigRootDir": "C:\\Users\\(...)\di"} was found.  
This is probably a typing mistake. Fixing it will remove this message.
- Validation Error: Preset ts-jest/presets/js-with-babel not found.

Other issues:

The test file:

- 'React' is defined but never used.eslint@typescript-eslint/no-unused-vars
- This assertion is unnecessary since it does not change the type of the expression.eslint@typescript-eslint/no-unnecessary-type-assertion
- 'expect' is not defined.eslintno-undef
- Unsafe member access .toBeInTheDocument on an `any` value.eslint@typescript-eslint/no-unsafe-member-access

Jest.config.cjs:

```
```
```

Parsing error: ESLint was configured to run on `/jest.config.cjs` using `parserOptions.project`:

- `<tsconfigRootDir>/tsconfig.json`
- `<tsconfigRootDir>/tsconfig.node.json`

```
```
```

jest.config.cjs file:

```
```
```

```
module.exports = {
  preset: 'ts-jest/presets/js-with-babel',
  testEnvironment: 'jsdom',
  transform: {
    '^.+\\.jsx?$': 'babel-jest',
  },
  parserOptions: {
    tsconfigRootDir: __dirname,
  },
};
```
```

### Vastaus 3:

It seems like there are a few issues to address. Let's go through them step by step:

#### ### 1. Jest Configuration Error:

The error "Unknown option 'parserOptions'" suggests that Jest does not recognize the `parserOptions` in your configuration. This might be due to an outdated version of Jest or ts-jest.

Update your Jest and ts-jest packages to the latest versions:

```
```bash
yarn add --dev jest ts-jest
```
```

### ### 2. ESLint Issues in Test File:

#### 'React' is defined but never used:

This is a common ESLint warning. It's safe to ignore if you're not using `React` explicitly in your test file. If you are using it, make sure it's imported where needed.

#### This assertion is unnecessary since it does not change the type of the expression:

Remove unnecessary type assertions. In your test file, make sure that type assertions are only used when necessary. For example, if you are asserting types for the `screen.getByText` result, ensure that it's required.

#### 'expect' is not defined:

Make sure you have imported `expect` from Jest at the beginning of your test file:

```
```javascript
import { expect } from '@jest/globals';
```
```

#### Unsafe member access `.toBeInTheDocument` on an `any` value:

If you are using `as` for type assertion, ensure that you are asserting to the correct type. If you're confident in the type, you can add a comment to satisfy ESLint:

```
```javascript
const buttonElement = screen.getByText(/click me/i) as HTMLElement; // es-
lint-disable-line @typescript-eslint/no-unsafe-member-access
```
```

### ### 3. Jest Configuration in `jest.config.cjs`:

The error "ESLint was configured to run on ``<tsconfigRootDir>/jest.config.cjs`" indicates a problem with the ESLint configuration. It seems like ESLint is configured to run on your Jest config file. Update your ESLint configuration to exclude Jest config files.`

Add the following to your ``.eslintrc.js``:

```
```javascript
module.exports = {
  // ... other configurations
  ignorePatterns: ['jest.config.cjs'],
};
```
```

This will instruct ESLint to ignore the Jest configuration file.

After making these changes, try running your tests again using ``yarn test``. Adjust the solutions based on your project's specific setup and requirements.

## LIITE E: GOOGLE BARD -KESKUSTELU TESTIYMPÄRISTÖN JA TESTIEN LUOMISEEN

### Syöte 1:

I need to create tests for the React project. The project uses Vite as build tool and contentful-cli to fetch data from Contentful. Yarn works as a package manager.

What I need to create tests?

---

### Vastaus 1:

To create tests for your React project with Vite, contentful-cli, and Yarn, you'll need the following tools and dependencies:

#### **Testing tools:**

- **Jest:** A popular JavaScript testing framework that can be used to test React components and applications.

- **React Testing Library:** A library that provides testing utilities specifically for React components and applications.

#### **Contentful data fetching:**

- **contentful-cli:** A command-line tool that allows you to manage and interact with your Contentful space.

- **contentful-management:** A Node.js library that provides a programmatic interface for managing your Contentful space.

#### **Package management:**

- **Yarn:** A package manager that can be used to install and manage JavaScript dependencies.

#### **Steps to create tests:**

##### 1. **Install Jest and React Testing Library:**

```
```bash
yarn add jest @testing-library/react
```
```

##### 2. **Configure Jest:**

Create a `jest.config.js` file in the root of your project and add the following configuration:

```
```javascript
module.exports = {
  preset: 'ts-jest',
  testEnvironment: 'jsdom',
}
```

```
  setupFilesAfterEnv: ['@testing-library/jest-dom/extend-expect'],
};
```
```

### 3. **Create test files:**

For each React component you want to test, create a corresponding test file with the same name as the component, but with the `.test.js` or `.test.tsx` extension.

### 4. **Write test cases:**

Inside each test file, import the React Testing Library and the component you're testing. Use Jest's `it` blocks to define test cases.

### 5. **Mock Contentful data:**

Since your project uses `contentful-cli` to fetch data from Contentful, you'll need to mock the data in your tests. Use Jest's `mock` function to create mock responses for your Contentful API calls.

### 6. **Run tests:**

To run your tests, use the following command:

```
```bash
yarn test
```
```

### 7. **Integrate with Vite:**

Vite doesn't automatically run Jest tests during development. To integrate Jest with Vite, you can use the `vite-jest` plugin. Install the plugin:

```
```bash
yarn add vite-jest
```
```

Then add the plugin to your Vite configuration file (usually `vite.config.js`):

```
```javascript
import { defineConfig } from 'vite';
import { viteJest } from 'vite-jest';

export default defineConfig({
  plugins: [viteJest()],
});
```
```

Now, Jest tests will be automatically run when you start Vite in development mode.

## Syöte 2:

Generate test for `Cards.tsx` file:

```

...
import { useEffect, useState } from 'react';
import fetchCards from '../..../functions/fetchCards';
import { TypeCardsFields } from '../..../contentful';

import './Cards.scss';

function Cards() {
  const [data, setData] = useState<TypeCardsFields>();

  const fetchData = () => {
    fetchCards()
      .then((result) => {
        const data = result!;
        setData(data);
      })
      .catch((error) => {
        console.error(error);
      });
  };

  useEffect(() => {
    fetchData();
  }, []);

  const cards = data?.cards;

  return (
    <div className="cards">
      {cards?.map((card) => {
        return (
          <div
            className="card"
            key={card.fields.title}
            style={{
              background:
                `url(${card.fields.image?.fields.file.url}) no-repeat center`,
            }}
          >
            {card.fields.title && card.fields.description && (
              <div className="card-inner">
                <div className="card-title">
                  {card.fields.showTitle && (
                    <h1>{card.fields.title}</h1>
                  )}
                </div>
                <div className="card-text">
                  <p>{card.fields.description}</p>
                </div>
              </div>
            )}
          </div>
        );
      })}
    </div>
  );
}

export default Cards;

```

...

## Vastaus 2:

Sure, here is the test file for the `Cards.tsx` file:

```

````tsx
import React from 'react';
import { render, screen } from '@testing-library/react';
import Cards from './Cards';
import { TypeCardsFields } from '../contentful';

jest.mock('../functions/fetchCards');

describe('Cards component', () => {
  it('should render the cards', async () => {
    const mockCardsData: TypeCardsFields = {
      cards: [
        {
          fields: {
            title: 'Card 1',
            description: 'This is the first card.',
            image: {
              fields: {
                file: {
                  url: 'https://example.com/image1.jpg',
                },
              },
            },
          },
        },
        {
          fields: {
            title: 'Card 2',
            description: 'This is the second card.',
            image: {
              fields: {
                file: {
                  url: 'https://example.com/image2.jpg',
                },
              },
            },
          },
        },
      ],
    };

    fetchCards.mockResolvedValue(mockCardsData);

    render(<Cards />);

    const cards = screen.getAllByRole('card');
    expect(cards).toHaveLength(2);

    expect(cards[0]).toHaveStyle({
      backgroundImage: `url(https://example.com/image1.jpg) no-repeat center`,
    });
  });
});

```

```

    });

    expect(cards[1]).toHaveStyle({
      backgroundImage: `url(https://example.com/image2.jpg) no-repeat center`,
    });
  });
});
});

```

This test file will first mock the `fetchCards` function so that it always returns a specific set of card data. Then, it will render the `Cards` component and use the testing library to assert that the cards are rendered correctly.

---

### Syöte 3:

After testing, I found few problems which are listed below:

- jest.config.js file: Parsing error: ESLint was configured to run on ``<tsconfigRootDir>/jest.config.js`` using `parserOptions.project`
- Cards.test.tsx file: 'React' is defined but never used.
- Cards.test.tsx file: Cannot find name 'jest'
- Cards.test.tsx file: Cannot find name 'describe'. Do you need to install type definitions for a test runner?
- Cards.test.tsx file: Cannot find name 'it'. Do you need to install type definitions for a test runner?
- Cards.test.tsx file: sync arrow function has no 'await' expression.
- Cards.test.tsx file: Type `{ url: string; }` is missing the following properties from type `{ url: string; details: { size: number; image?: { width: number; height: number; } | undefined; }; fileName: string; contentType: string; }: details, fileName, contentType`
- Cards.test.tsx file: Unsafe member access `.mockResolvedValue` on an `any` value.
- Cards.test.tsx file: Unsafe member access `.toHaveLength` on an `any` value.
- Cards.test.tsx file: Unsafe member access `.toHaveStyle` on an `any` value.
- vite.config.ts file: This rule requires the `strictNullChecks` compiler option to be turned on to function correctly.
- vite.config.ts file: Cannot find module 'vite-jest' or its corresponding type declarations.
- vite.config.ts file: Unsafe assignment of type `any[]` to a variable of type `PluginOption[]`.

---

### Vastaus 3:

Sure, I can help you with these problems.

**\*\*jest.config.js file:\*\***

The error message is telling you that ESLint is configured to run on the `jest.config.js` file using the `parserOptions.project` option. This means that ESLint is expecting to find a `tsconfig.json` file in the root directory of the project. If you have not yet created a `tsconfig.json` file, you can do so by running the following command:

```

```bash
npx tsc --init

```

```
...
```

This will create a basic `tsconfig.json` file that you can then customize as needed.

**\*\*Cards.test.tsx file:\*\***

The errors in the `Cards.test.tsx` file are all related to the fact that you have not installed the type definitions for Jest. To install the type definitions for Jest, you can run the following command:

```
```bash
yarn add @types/jest
```
```

This will install the `@types/jest` package, which will provide the type definitions for Jest to the TypeScript compiler.

**\*\*vite.config.ts file:\*\***

The errors in the `vite.config.ts` file are related to the fact that you have not installed the `vite-jest` plugin. To install the `vite-jest` plugin, you can run the following command:

```
```bash
yarn add vite-jest
```
```

This will install the `vite-jest` package, which will provide the plugin for integrating Jest with Vite.

Once you have installed all of the necessary dependencies, you should be able to run your tests without any errors.

Here is a summary of the steps to fix the problems you listed:

1. Create a `tsconfig.json` file in the root directory of the project.
2. Install the `@types/jest` package.
3. Install the `vite-jest` plugin.

I hope this helps!

## LIITE F: GITHUB COPILOT -KESKUSTELU TESTIYMPÄRISTÖN JA TESTIEN LUOMISEEN

### Syöte 1:

I need to create tests for the React project. The project uses Vite as build tool and contentful-cli to fetch data from Contentful. Yarn works as a package manager.

What I need to create tests?

---

### Vastaus 1:

GitHub Copilot: To create tests for your React project, you will need a testing framework and assertion library. Here are the steps you need to follow:

1. Install Jest as your testing framework. Jest is widely used for testing React applications.
2. Install React Testing Library. This library is a very light-weight solution for testing React components. It provides light utility functions on top of `react-dom` and `react-dom/test-utils`, in a way that encourages better testing practices.
3. Install `@testing-library/jest-dom` to provide custom jest matchers for asserting on DOM nodes.
4. Install `@testing-library/user-event` to simulate user events on DOM elements.
5. Install `vite-plugin-testing-library` to integrate testing-library with Vite.
6. Update your `package.json` to include a script for running your tests.

Here is the code to install these packages:

```
```bash
yarn add --dev jest @testing-library/react @testing-library/jest-dom @testing-library/user-event vite-plugin-testing-library
```
```

And update your `package.json` scripts:

```
```json
"scripts": {
  "test": "jest"
}
```

You will also need to create a configuration file for Jest (`jest.config.js`) and setup your tests. You can create a `\_\_tests\_\_` directory in your project root and start writing your tests there.

For testing components that fetch data from Contentful, you might want to consider using a library like `msw` (Mock Service Worker) to intercept these requests and provide mock responses during testing. This allows you to test your components in isolation without needing to hit the actual Contentful API.

---

## Syöte 2:

I tried to install packages, but I got this error message:

```

...
yarn add v1.22.19
[1/4] Resolving packages...
error An unexpected error occurred: "https://registry.yarnpkg.com/vite-plugin-testing-library: Not found".
info If you think this is a bug, please open a bug report with the information provided in "C:\\Users\\markus.palomaki\\Documents\\dippa\\di\\yarn-error.log".
info Visit https://yarnpkg.com/en/docs/cli/add for documentation about this command.
...

```

Also, what I should add into `jest.config.js` file

---

## Vastaus 2:

GitHub Copilot: I apologize for the confusion, there seems to be a mistake in my previous message. There's no package named `vite-plugin-testing-library`. You can remove it from the installation command.

Regarding the `jest.config.js` file, here's a basic setup:

```

```javascript
module.exports = {
  roots: ['<rootDir>/src'],
  testMatch: ['**/__tests__/**/*.+(ts|tsx|js)',
    '**/?(*.)+(spec|test).+(ts|tsx|js)'],
  transform: {
    '^.+\\.?(ts|tsx)$': 'ts-jest',
  },
  setupFilesAfterEnv: ['@testing-library/jest-dom/extend-expect'], // setupFiles before the tests are ran
};
```

```

This configuration sets the root directory for Jest to look for tests in the `src` directory. It also specifies that Jest should look for test files with a `.spec` or `.test` suffix in the `\_\_tests\_\_` directory (or anywhere in the project).

The `transform` option is used to tell Jest that `ts` and `tsx` files should be transformed using `ts-jest`.

The `setupFilesAfterEnv` option is an array of paths to modules that run some code to configure or set up the testing framework before each test. In this case, it's setting up `jest-dom` for each test.

Please run the corrected installation command:

```

```bash
yarn add --dev jest @testing-library/react @testing-library/jest-dom @testing-library/user-event
```

```

---

### Syöte 3:

Create Jest tests for the Cards.tsx file.

---

### Vastaus 3:

GitHub Copilot: To create Jest tests for the `Cards.tsx` file, you will need to mock the `fetchCards` function and simulate its response. Here's a step-by-step plan:

1. Import the necessary libraries and the component to be tested.
2. Mock the `fetchCards` function.
3. Write a test to check if the component renders without crashing.
4. Write a test to check if the `fetchCards` function is called on component mount.
5. Write a test to check if the cards are rendered correctly based on the mock data.

Here's the code for the tests:

```

```javascript
import { render, screen, waitFor } from '@testing-library/react';
import { rest } from 'msw';
import { setupServer } from 'msw/node';
import Cards from '../path/to/Cards';
import fetchCards from '../..../functions/fetchCards';

jest.mock('../..../functions/fetchCards');

const mockData = {
  cards: [
    {
      fields: {
        title: 'Test Card 1',
        description: 'Test Description 1',
        image: {
          fields: {
            file: {
              url: 'test-url-1',
            },
          },
        },
      },
    },
  ],
};
// Add more cards as needed
],

```

```
};

beforeEach(() => {
  (fetchCards as jest.Mock).mockResolvedValue(mockData);
});

test('renders without crashing', () => {
  render(<Cards />);
});

test('calls fetchCards on mount', async () => {
  render(<Cards />);
  await waitFor(() => expect(fetchCards).toHaveBeenCalledTimes(1));
});

test('renders cards correctly', async () => {
  render(<Cards />);
  await waitFor(() => screen.getByText('Test Card 1'));
  expect(screen.getByText('Test Card 1')).toBeInTheDocument();
  expect(screen.getByText('Test Description 1')).toBeInTheDocument();
});
```
```

Please replace ``../path/to/Cards`` with the actual path to the `Cards.tsx` file in your project.