

# CHATGPT JA COPILOT OHJELMOINNIN TYÖKALUINA

Ylihoikka Peter

Opinnäytetyö

Tieto- ja viestintäteknikka  
Insinööri (AMK)

2024

Tieto- ja viestintäteknikka  
Insinööri (AMK)

---

<b>Tekijä</b>	Peter Ylihoikka	<b>Vuosi</b>	2024
<b>Ohjaaja</b>	Petri Hannula		
<b>Toimeksiantaja</b>	FrostBit Software Lab		
<b>Työn nimi</b>	ChatGPT ja Copilot ohjelmoinnin työkaluna		
<b>Sivumäärä</b>	34		

---

Tämän opinnäytetyön tavoitteena oli tekoälyjen ChatGPT:n ja Copilotin tutkiminen ohjelmoinnin työkaluina sekä kartoittaa tekoälyjen tämänhetkistä kyvykkyyttä ohjelmoinnissa Python-kielellä. Opinnäytetyön toimeksiantajana oli FrostBit Software Lab -ohjelmistolaboratorio, joka on osa Lapin ammattikorkeakoulua.

Opinnäytetyön tietoperusta on koottu kirjallisuudesta, tiedeartikkeleista ja dokumentaatiosta, joiden avulla käydään läpi tekoälyjen peruskäsite, kehitys ja ominaisuudet. Tekoälyjen OpenAI ChatGPT:n ja Github Copilotin ominaisuuksia ja kehitystä kerrotaan dokumentaation avulla. Tässä työssä tehtiin myös ChatGPT- ja Copilot-tekoälyjen testaamista eritasoisilla ohjelmointitehtävillä, joilla mitattiin niiden kykyä generoida ohjelmakoodia pyynnöistä.

Tutkimusosion tuloksista voidaan päätellä, että tekoälyillä on pysyvä paikka ohjelmoinnin piirissä työkaluna. Tekoälyt pystyvät generoimaan pienistä keskisuuriin kokonaisuuksiin ratkaisuja ripeästi ja nopeuttavat ohjelmistokehittäjien työkulkuja huomattavasti. Tekoäly ei ole kuitenkaan ole syrjäyttämässä ihmisiä ohjelmoinnissa, koska niiden tehokas käyttö työtehtävissä vaatii hyvän perusosaamisen ja ymmärryksen ohjelmoinnista.

Tekoälyjen integrointia ohjelmointiympäristöihin työkaluina voi suositella aloittelijoille sekä kokeneille ohjelmistokehittäjille. Tekoälyistä ohjelmointiin Copilot soveltuu paremmin työkaluksi, koska sen käytön nopeus ja generoitujen vastauksien täsmällisyys oli parempi kuin ChatGPT. Tutkimusosio toteutettiin hyödyntämällä ilmaisia kokeiluversioita palveluista ChatGPT ja Copilot.

Degree programme in Information  
and Communication Technology  
Bachelor of Engineering

---

<b>Author</b>	Peter Ylihoikka	<b>Year</b>	2024
<b>Supervisor</b>	Petri Hannula		
<b>Commissioned by</b>	FrostBit Software Lab		
<b>Title</b>	ChatGPT and Copilot as tools in programming		
<b>Number of pages</b>	34		

---

The aim of this thesis was to study the artificial intelligences ChatGPT and Copilot as tools in programming, and to map the current abilities of artificial intelligences in programming in the Python language. The commissioner of the thesis was FrostBit Software Lab, which is a part of Lapland University of Applied Sciences.

The knowledge base of the thesis was compiled from literature, scientific articles, and documentation, which were used to review the basics, development, and characteristics of artificial intelligence. The features and development of the artificial intelligences OpenAI ChatGPT and GitHub Copilot were explained with the help of documentation. The focus of this thesis study was to test the abilities of ChatGPT and Copilot in programming tasks of different levels to measure their ability to generate program code in the Python language.

From the results of the research section of this thesis, it can be concluded that artificial intelligence has a semi-permanent place within programming. Artificial intelligence can generate solutions for small- to medium-sized tasks quickly, significantly speeding up the workflows of software developers. The integration of artificial intelligence into software development environments can be recommended for both beginners and experienced software developers. The research section was conducted by utilizing free trial versions of the services ChatGPT and Copilot.

Keywords

artificial intelligence, ChatGPT, Copilot, programming

# SISÄLLYS

1 JOHDANTO.....	5
2 KATSAUS TEKOÄLYYN .....	7
2.1    Tekoälyn historia ja kehitys.....	7
2.2    Tekoälyt apuna ohjelmoinnissa.....	8
2.2.1    OpenAI ChatGPT .....	9
2.2.2    Github Copilot.....	10
2.3    Eettiset haasteet.....	12
3 OHJELMOINTITEHTÄVÄT .....	13
3.1    GhatGPT-ohjelmointitehtävät.....	14
3.1.1    Testi 1: Numeropeli .....	14
3.1.2    Testi 2: Virheiden korjaus annetusta koodista.....	16
3.1.3    Testi 3: Ristinollapeli.....	17
3.2    Copilot-ohjelmointitehtävät .....	20
3.2.1    Testi 1: Numeropeli .....	21
3.2.2    Testi 2: Virheiden korjaus annetusta koodista.....	22
3.2.3    Testi 3: Ristinollapeli.....	24
3.3    Tuloksien vertailu ja analyysi .....	28
4 POHDINTA.....	31
LÄHTEET .....	33

## 1 JOHDANTO

Tekoäly eli AI (artificial intelligence) on noussut tieteenalana valtavaksi kehityksen kentäksi, jossa on tapahtunut suuria harppauksia viimeisen kymmenen vuoden aikana. Tekoälyä integroidaan nykyään yhä useampaan toimialaan ja ammattiin, joissa se usein esiintyy median tai ohjelmiston muodossa.

Tämän opinnäytetyön ensisijaisena tavoitteena on antaa kattava käsitys tekoälyn integroimisesta ohjelmoinnin työkaluna. Opinnäytetyö keskittyy ohjelmoinnin piiriin tarkastelemalla tekoälyjen integrointia työkaluina osaksi kehitysprosessia. Se alkaa ohjelmoinnin nykytilan ja tekoälyn kasvavan roolin tutkimisesta nykypäivänä.

Tutkimus keskittyy tunnistamaan tekoälyjen hyötyjä perinteisissä ohjelmointimenetelmissä ja luomaan perustan tarpeelle integroida tekoäly osaksi ohjelmointityökaluja. Kirjallisuuskatsaukseen pohjautuva tietopohjan luonti tarjoaa kontekstin tekoälylle ohjelmoinnissa ja seuraa sen kehitystä.

Työ analysoi AI:n sisällyttämisen hyödyt, kuten lisääntyneen tuottavuuden, parannetun koodin laadun ja toistuvien tehtävien automatisoinnin. Lisäksi katsauksessa tarkastellaan haasteita, jotka liittyvät tekoälyjen integroimiseen osaksi ohjelmointia.

Tavoitteena on esittää mahdolliset hyödyt, joita tekoäly tuo ohjelmointiprosessille sekä samalla käsitellä sen toteuttamiseen liittyviä haasteita ja rajoituksia. Opinnäytetyön sisältö soveltuu aiheesta kiinnostuneille ja tekoälyn hyödyntämistä työkaluna ohjelmistokehityksessä tutkiville.

Lopuksi opinnäytetyö pyrkii olemaan resurssi tekoälyn ja ohjelmoinnin välisen suhteen ymmärtämisessä. Käsittelemällä tutkimuskysymyksiä ja -tavoitteita ohjelmointitehtävien muodossa työ pyrkii antamaan katsauksen meneillään olevaan vuoropuheluun tekoälyn integroimisesta muuntavana työkaluna ohjelmointiympäristöön.

Tämän tutkimuksen ensisijaiset tavoitteet ovat tunnistaa haasteet ja rajoitukset, jotka liittyvät AI-työkalujen käyttämiseen ohjelmoinnissa. Tähän kuuluu tekoälyn

luoman koodin mahdollisten harhojen tutkiminen, eettiset näkökohdat ja esteet, joita kehittäjät kohtaavat integroidessaan näitä työkaluja työkulkuihinsa.

Arvioimalla ChatGPT:n ja Copilotin ominaisuuksia ja rajoituksia vertailussa pyritään ymmärtämään niiden vaikutus tuottavuuteen, koodin laatuun ja kehittäjäkokemukseen. Näihin tavoitteisiin pyrkimällä tutkimuksella pyritään lisäämään ymmärrystä tekoälytyökalujen roolista ohjelmoinnissa ja tarjoamaan näkemyksiä niiden mahdollisuuksista nopeuttaa ohjelmistokehitysprosessia.

Ohjelmointitehtävien avulla analysoidaan tekoälyjen OpenAI ChatGPT:n sekä Github Copilotin kykyä suoriutua ohjelmointitehtävistä, kuten ohjelmakoodin generointi, virheenkorjaus sekä ohjelmoinnin avustaminen. Vertaillaan näiden työkalujen täsmällisyyttä, nopeutta ja helppokäyttöisyyttä Visual Studio Code - ohjelmointiympäristössä.

Tutkimuskysymykset voidaan tiivistää kolmeen pää-kysymykseen:

1. Kuinka hyvin tekoälyt ChatGPT ja Copilot kykenevät suoriutumaan ohjelmointipainotteisista tehtävistä?
2. Missä vaiheessa tekoälyt ovat ja tarviiko kehittäjien muokata tekoälyjen generoimia vastauksia?
3. Kuinka helppoa on integroida tekoälyt osaksi ohjelmoinnin työkaluja?

Tutkimuskysymyksien tavoitena on tarjota suosituksia käyttäjille, kuinka AI-työkaluja ChatGPT ja Copilot voidaan parhaiten hyödyntää ohjelmointikäytännöissään. Tämä voi sisältää työkalujen käyttöönoton, hyviä käytäntöjä, integrointia kehitysympäristöihin ja strategioita tekoälyavun hyötyjen maksimoimiseksi. Tutkitaan myös ChatGPT:n ja Copilotin käytön vaikutusta kehittäjien tuottavuuteen ja koodin laatuun.

## 2 KATSAUS TEKOÄLYYN

Tekoälyllä viitataan ihmisälyn simulointiin tietokoneilla. Tekoälyt on kehitetty oppimaan niille annetusta datasta, joista ne kykenevät tunnistamaan kuvioita ja suorittaa tehtäviä, jotka tyypillisesti vaativat ihmisen älykkyyttä. Tekoälyjen teknologioihin kuuluvat koneoppiminen, kielten prosessointi ja konenäkö. Tekoälyllä on lukuisia käyttökohteita monilla eri aloilla ja kehittyä jatkuvasti. (IBM 2023.)

### 2.1 Tekoälyn historia ja kehitys

Tieteenalana tekoäly sai alkunsa 1940-luvulla, kun tietokoneiden kehityksen myötä tutkijat halusivat selvittää olisiko saada koneet ajattelemaan ihmisen tavoin (Ojanperä 2023, 24). Tekoälystä löytyy mainintoja antiikin kreikan kirjallisuudesta, jossa pohdittiin konseptia älyllisestä koneesta (Shashkevich 2019).

Vuonna 1956 kesäkuussa järjestettiin alan tutkijoille seminaari nimellä The Dartmouth Summer Research Project on Artificial Intelligence, joka kesti kokonaisuudessaan kahdeksan viikkoa. Tämän seminaarin järjestäjät olivat tutkijat Marvin Minsky, John McCarthy, Claude Shannon ja Nathaniel Rochester. Seminaarissa esiteltiin yksi ensimmäisistä tekoälyohjelmista, jonka toiminta perustui pitkälti ennalta ohjelmoituun sääntöpohjaiseen toimintaan. (Ojanperä 2023, 24.)

Kehitys hidastui 1960-luvun alussa, kun huomattiin kuinka rajoitetusti koneet pystyivät tekemään päätöksiä loogisella sääntöpohjaisella kaavalla. Kunnes 1980-luvulla kehitykseen tuli käänne neuroverkkojen ja syväoppimisen myötä, joiden avulla koneet kykenisivät teoriassa ohjelmistokehittäjien määräämien tehtävien lisäksi oppimaan ja paranemaan ajan myötä. (Ojanperä 2023, 24–25.)

Neuroverkot simuloivat ihmisten neuroniverkkojen toimintaa aivoissa yhdistämällä suuren määrän laskentakaavoja yhdeksi suureksi "neuroverkoksi", jossa laskentakaavat ovat koneen aivojen neuroneja. Näiden yhdistettyjen kaavojen avulla voidaan opettaa koneita tunnistamaan kuvia, ääniä, tekstejä ja muuta dataa niiden ominaispiirteistä. Syväoppiminen eli "deep learning"

tarkoittaa neuroverkkojen soveltamista datan tunnistamiseen käyttäen monikerroksisia verkkoja, joissa jokainen kerros oppii tarkempia piirteitä sille syötetystä datasta. (Salo 2023, 14–15.)

Tekoälyn nopea kehitys 2010-luvun jälkeen voidaan tiivistää pariin huomattavaan muuttujaan, jotka ovat vaikuttaneet eniten. Ensimmäinen muuttuja on tietokoneiden fyysiset komponentit eli ”hardware”, jonka ansiosta koneiden laskentateho on moninkertaistunut lyhyessä ajassa. Toisena muuttujana on vapaasti saatavilla oleva datan määrä verkossa, joka on maailman digitalisaation myötä johtanut datamääriin helposti saatavilla olevaan tietoon. (Kananen & Puolitaival 2019, 35–36.)

## 2.2 Tekoälyt apuna ohjelmoinnissa

Varhaiset yritykset soveltaa tekoälyä ohjelmointiin olivat pitkälti sääntöpohjaisia asiantuntijajärjestelmiä, jotka tarjosivat vain vähän apua sellaisissa tehtävissä kuin koodin luominen ja virheiden korjaus (Ojanperä 2023, 24). Kaksi merkittävää AI-työkalujen kehittäjää ovat OpenAI:n ChatGPT ja Githubin Copilot, joiden kehittämät palvelut ovat laajassa käytössä (Salo 2023, 32–34).

Ohjelmiston kehityksessä on tapa debugaamiseen, jota kutsutaan ”rubber duck debugging” -termillä. Tällä tarkoitetaan mahdollisten ongelmakohtien ratkointaa puhumalla ääneen tai kirjoittamalla toiselle osapuolelle, joka on perinteisesti ollut fyysinen tai fiktiivinen kumiankka. Tarkoituksena on käydä ohjelmakoodi rivi riviltä läpi ja selittää sen toiselle osapuolelle niin, että sen toiminta on helppo ymmärtää. Tekoälyt soveltuvat todella hyvin tähän tarkoitukseen, koska ne pystyvät vastaamaan takaisin ja selittämään ohjelmakoodin rakenteen sekä tarjota useita mahdollisia ratkaisuja kehittäjälle pyydettyäessä. (Shaw 2023.)

Tekoälyn käyttäminen työkaluna ohjelmoinnissa vaatii silti kehittäjältä hyvää perustaitoa ja ymmärrystä, jotta osaa muotoilla pyynnön oikein tekoälylle tulkittavaksi. Tekoälyn generoima vastaus ei saata vastata haluttua lopputulosta, joten generoituun vastaukseen joutuu usein tekemään muutoksia. (Oguz 2023.)



### 2.2.1 OpenAI ChatGPT

ChatGPT on OpenAI:n kehittämä keskustelupohjainen tekoäly, joka julkaistiin vuonna 2022. Se perustuu GPT-arkkitehtuuriin, joka on lyhenne sanoista "Generative Pre-trained Transformer" (Ojanperä 2023, 28). ChatGPT voi ymmärtää ja luoda ihmisen kaltaista tekstiä vastaanottamansa syötteen perusteella. Se pystyy osallistumaan erilaisiin keskusteluihin, vastaamaan miltei mihin tahansa kysymyksiin (OpenAI 2022a). ChatGPT on koulutettu valtavilla tekstidatamäärillä ja se on erinomainen ymmärtämään ja luomaan ihmisen kaltaista tekstiä käyttäjien antamien kehotteiden perusteella (OpenAI 2022b).

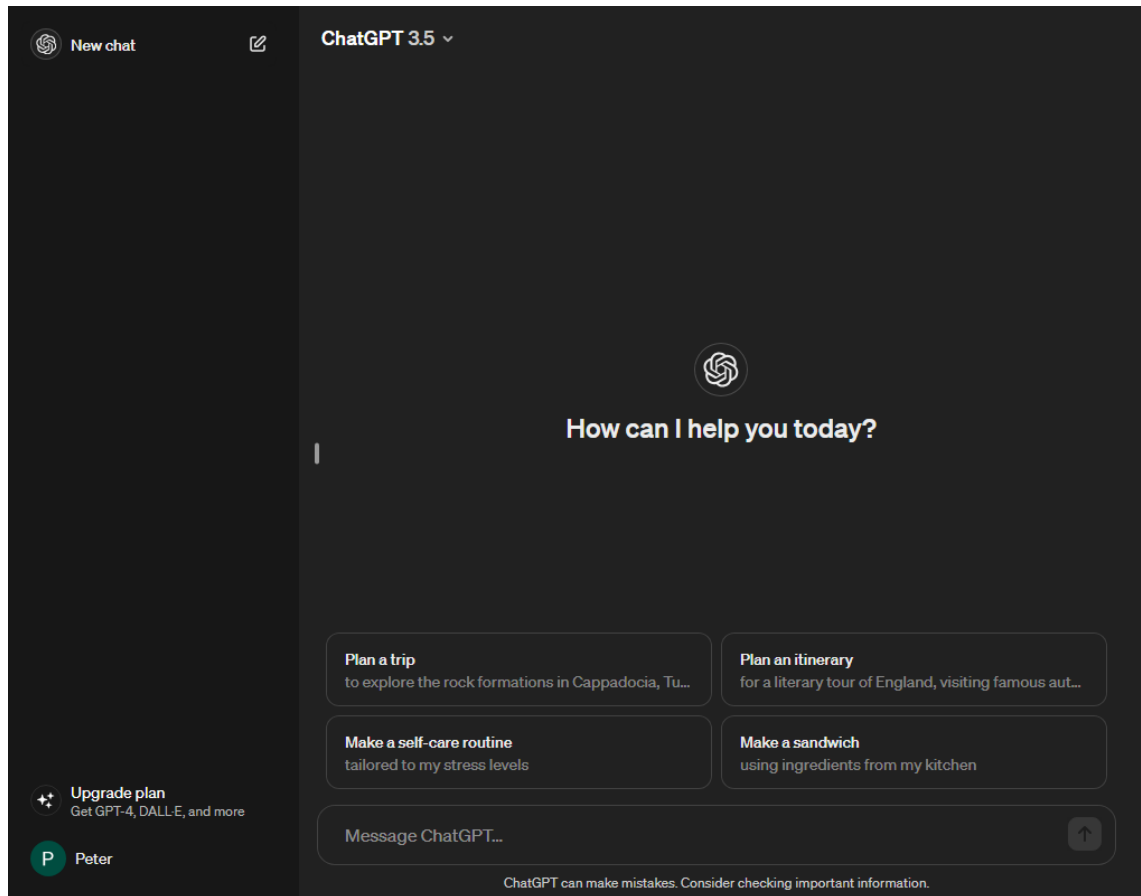
ChatGPT:n toiminta perustuu neuroverkkoihin ja syväoppimiseen, joiden avulla emuloidaan ihmisaivojen toimintaa. ChatGPT on koulutettu suurella datamäärällä, joka koostuu pääsääntöisesti ihmisten tuottamasta datasta, johon kuuluu kaikki erinlainen media kuten: kirjat, dokumentit, raportit, artikkelit useista eri aiheista ja lähteistä. ChatGPT on koulutettu vahvistusoppimista hyödyntäen ihmisten avulla (RLHF, Reinforcement Learning from Human Feedback). Ensimmäiset kielimallit kehitettiin syöttämällä tekoälylle keskusteluja, joissa oli ihminen vastaamassa molemmin puolin dialogia. (Salo 2023, 43.)

ChatGPT vaikuttaa todella älykkäältä ja tehokkaalta, mutta se on silti altis virheille, koska kaikki sen generoimat vastaukset ovat tekstin ennustamista. Ennustaminen perustuu mallien parametreihin, jotka on muovattu vahvistusoppimisella ja opetusaineistolla. Virheitä voi esiintyä valheellisten taikka järjettömien vastauksien muodossa. (Salo 2023, 43.)

ChatGPT:n käyttöehdoissa on muutamia tärkeitä OpenAI:n määrittelemiä asioita, jotka on hyvä tietää. ChatGPT hyödyntää pääasiassa koneoppimista ja jolloin käyttäjä saa myös omistusoikeuden ChatGPT:n generoimaan sisältöön. OpenAI voi myös kerätä palveluihinsa syötettyä sisältöä tekoälyjen kehittämiseen, ellei käyttäjä kiellä sitä erikseen. (Salo 2023, 45.)

Ilmainen GPT-3.5 -kielimalli on saatavilla osoitteessa (<https://chatgpt.com/>) verkkosivun käyttöliittymän (kuvio 1) kautta, ja ChatGPT on käyttövalmis heti tunnuksien luomisen ja kirjautumisen jälkeen. Kehittyneemmän GPT-4 -kielimallin käyttö vaatii maksullisen 20 dollarin "Plus"- tai 30 dollarin "Team" -

jäsenyyden. Lisäksi on "enterprise" -jäsenyys, joka on suunnattu yrityksille ja organisaatioille. Hintatiedot ovat haettu ChatGPT:n sivuilta tämän opinnäytetyön tekemisen hetkellä (26.5.2024), ja hinnat saavat vaihdella ajan myötä kun tekoälymallit kehittyvät ajan myötä. (OpenAI 2022c.)



Kuvio 1. ChatGPT:n verkkosivun käyttöliittymä

### 2.2.2 Github Copilot

Copilot, jonka GitHub on kehittänyt yhteistyössä OpenAI:n kanssa. Copilot julkaistiin vuonna 2021 hyödyntäen GPT-teknologiaa, kuten ChatGPT Copilot on koulutettu useille eri ohjelmointikielille ja julkisesti saatavilla oleville ohjelmistokoodille. (Github 2023).

Copilot hyödyntää OpenAI:n kehittämää GPT-4-kielimallia, jonka avulla se tuo tarkempia ehdotuksia ja selityksiä ohjelmointiin liittyen. Sovellus pystyy antamaan reaaliajassa ehdotuksia, etsimään virheitä, parantamaan ohjelmakoodin laatua sekä nopeuttamaan työskentelyä. Copilot käyttää kaikkia

avoinna olevia sivuja ohjelmakoodieditorissa, jotta se pystyy paremmin ymmärtämään sille annettujen pyyntöjen kontekstia. (Github 2023.)

Copilot vaatii Github-käyttäjätilin ja Copilot-tilauksen, jonka käyttäminen maksaa normaalisti kymmenen euroa kuussa tai sata euroa vuodessa. Copilotista on saatavilla myös kolmenkymmenen päivän koeaika, jotta Copilottia voi kokeilla maksutta. Tämä vaatii rekisteröitymisen sekä maksutavan lisäämisen Github-tiliin. Hintatiedot haettiin Copilotin verkkosivuilta (26.5.2024) opinnäytetyön tekohetkellä. (Github 2022a.)

Copilotin asentaminen Visual Studio Code -ympäristöön on todella helppoa ja nopeaa. Laajennus ladataan Visual Studio Marketplace -sivulta, ja se asentuu itsestään. Seuraavan käynnistyksen yhteydessä Copilot on käyttövalmis ja sen tilan näkee Visual Studio Coden sisältä oikeasta alareunasta, jossa sijaitsee pieni ikoni. Copilot-ohjelmointitehtävät suoritetaan Visual Studio Code -ohjelmointiympäristössä, johon asennetaan Github Copilot -laajennus.

Kun Copilot on asennettu yhteen sen tukemista ohjelmointiympäristöistä joihin kuuluu: VS Code, Neovim, Visual Studio tai JetBrains IDEs, voidaan käyttää CLI-käyttöliittymää, joka on lyhenne sanoista "chat-like interface". CLI on ohjelmointiympäristön sisäinen chat-ikkuna, johon voi syöttää komentoja ja pyyntöjä sanallisesti Copilotin tulkittavaksi painamalla näppäinyhdistelmää "ctrl + i". Copilotin CLI on koulutettu pääsääntöisesti englanniksi ja käyttäjiä kehoitetaan tekemään pyynnöt englanniksi. (Github 2023.)

Saatujen vastauksien tarkkuus voi vaihdella monista eri tekijöistä, kuten: käyttöjärjestelmä, syötetty data ja avoinna olevat välilehdet ohjelmointiympäristössä, joita hyödyntäen Copilot generoi vastauksiaan. (Github 2023).

Copilot sisältää erilaisia komentoja, joilla voidaan pyytää tekoälyä suorittamaan niitä vastaavia tehtäviä ohjelmistoympäristön sisällä. Näihin komentoihin kuuluu "/fix", jonka avulla tekoäly skannaa valitun ohjelmakoodin läpi ja etsii mahdollisia virheitä ja generoi tarvittaessa paremman ratkaisun tilalle. Copilot pystyy myös selittämään kokonaisuudessaan valitun ohjelmakoodin "/explain" -komennolla, joka on todella hyödyllinen ohjelmakoodin ymmärtämisen apuna. (Github 2023.)

Ohjelmakoodin dokumentaatiota ja kommentointia varten on olemassa komento `"/doc"`, joka generoi valinnalle kommentit ohjelmakoodin sisään. Koodin kommentointi kuuluu hyviin ohjelmointikäytänteisiin, koska se auttaa itse ohjelmakoodin kaikkia, jotka työskentelevät kommentoidun ohjelmakoodin parissa ymmärtämään ohjelmakoodissa tapahtuu. (Github 2023.)

### 2.3 Eettiset haasteet

Tekoälyn kehittyessä on entistä tärkeämpää pohtia sen eettisiä seurauksia. Mitä enemmän tekoälyä aletaan hyödyntämään eri aloilla useissa eri tehtävissä, sitä suuremmaksi nousee tarve luoda sääntöjä ja eettisiä ohjeistuksia, joiden avulla tekoälyä voidaan käyttää vastuullisesti. (Salo 2023, 17.)

Etiikan aihepiiriin kuuluvat myös vaikutukset useisiin ammatteihin, joissa tekoälyä voidaan soveltaa. Tekoälyjen kehitys vaikuttaa myös talouteen ja tekoälyn vaikutus tulee väistämättä useisiin työpaikkoihin, joissa tekoäly saattaa syrjäyttää ihmisten hoitamat tehtävät ja palvelut. (Salo 2023, 18.)

Vaikutus akateemisissa piireissä esiintyy tekoälypalveluiden käytössä myös opinnoissa. Jos osalla oppilaista on kehittyneempiä tekoäly-palveluita käytössä he saattavat saada parempia arvosanoja kursseista verrattuna niihin, jotka eivät käytä tekoälyä apuna esimerkiksi ohjelmoinnissa. Palveluiden maksullisuus on myös otettava huomioon, koska kaikilla ei ole varaa maksaa useiden palveluiden kuukausimaksuja. Tästä syntyy eriarvoisuuksia akateemisissa sekä ammatillisissa ympäristöissä. (Salo 2023, luku 18–19.)

Tekoälyjen tekijänoikeudellisen suojan ympärillä on usein paljon keskustelua aiheesta, että voisivatko tekoälyjen generoima sisältö olla tekijänoikeuslailla suojattuja. Alalla ollaan kuitenkin samaa mieltä, että tuotokset ei voi nauttia tekijänoikeudellista suojaa, koska periaatteena vain ihminen kykenee luomaan omaperäisiä lopputuloksia ja teoksia. (Kallioniemi 2022, 123.)

### 3 OHJELMOINTITEHTÄVÄT

Tässä osiossa tavoitteena oli selvittää, kuinka hyvin tekoälyt suoriutuvat niille annetuista Python-kielisistä tehtävistä. Tehtäviä oli kolme eri luokkaa, joilla mitattiin niiden suoriutumista. Tehtävillä haettiin täsmällisyyttä ja toimivuutta tehtävänannon rajoissa.

Testausympäristönä toimi Visual Studio Code-ohjelmointiympäristö, jossa käännettiin ja testattiin tekoälyn generoimat ohjelmakoodit annettuihin tehtäviin. Ohjelmointikielenä testeissä oli Python.

Testi 1 oli yksinkertainen numeropeli, jossa tehtävänä oli luoda interaktiivinen arvauspeli. Tehtävällä haetaan eroavaisuuksia tekoälyjen generoimista ratkaisuista.

Testissä 2 mitattiin tekoälyjen kykyä löytää virheitä niille annetusta koodipätkästä ja niiden generoimien korjausehdotuksien laatua. Alla (kuvio 2) on testin 2 viallinen ohjelmakoodi. Se sisälsi kolme virhettä, joita ohjelmointiympäristöt, kuten VS Code ei havaise automaattisesti virheiksi. Virheet ovat hyvien ohjelmointikäytänteiden vastaisia. Ohjelma silti kääntyy ajettaessa normaalisti ja antaa oikean tuloksen.

1. "len(numbers)" käyttö "for" -silmukassa on huono tapa, joka tulisi korvata suoraan iteraatiolla "numbers"-taulukon yli.
2. Funktion "def sum\_of\_squares(numbers)" tulisi käsitellä tyhjää listaa oikein.

```
1 def sum_of_squares(numbers):
2     total = 0
3     for i in range(len(numbers)):
4         total += numbers[i] ** 2
5     return total
6
7 numbers = [1, 2, 3, 4, 5]
8 result = sum_of_squares(numbers)
9 print("Sum of squares:", result)
```

Kuvio 2. testin 2 viallinen ohjelmakoodi

Testi 3 oli ristinollapeli, jota pystyi pelata kaksi henkilöä ja tehtävän suorittaminen vaatii useamman funktion ja toistorakenteen, jolla saadaan lisää dataa kerättyä tekoälyjen generoimista ratkaisuista. Ratkaisu vaati myös funktion, jolla tarkistettiin arvoja listoista missä oli jo merkintöjä.

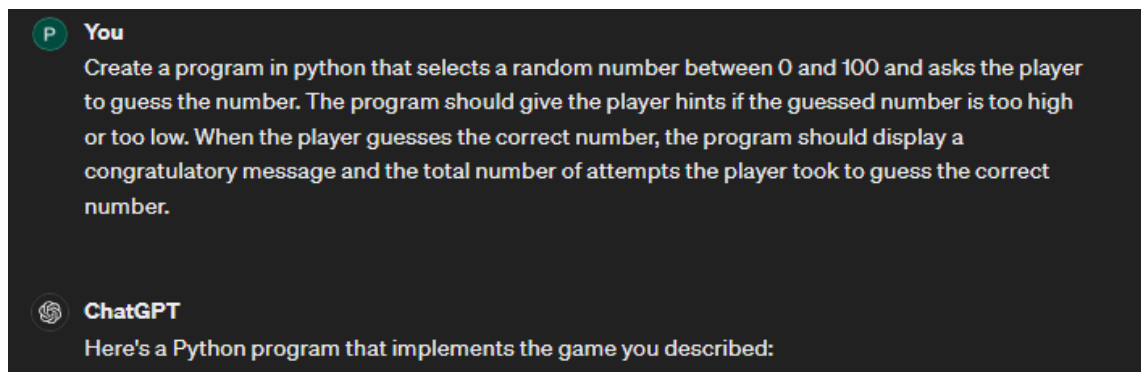
### 3.1 GhatGPT-ohjelmointitehtävät

ChatGPT:n ohjelmointitehtävät suoritettiin verkkoselaimen käyttöliittymän kautta, josta generoitu koodi kopioitiin annetussa muodossa Visual Studio Code -ohjelmointiympäristöön testausta varten ilman muokkauksia. AI:n generoimat ratkaisut tehtäviin testattiin Visual Studio Coden sisällä Python-tiedostona.

#### 3.1.1 Testi 1: Numeropeli

Testissä 1 tehtävänantona oli luoda ohjelma Python-kielellä, joka valitsee satunnaisluvun väliltä 0–100 ja pyytää pelaajaa arvaamaan luvun. Ohjelman tulee antaa pelaajalle vihjeitä, jos arvattu luku on liian suuri tai liian pieni. Kun pelaaja arvaa oikean luvun, ohjelman tulee näyttää onnitteluviesti. Pelin lopussa pelaajalle tulee näyttää, kuinka monta kertaa pelaaja on yrittänyt arvata oikeaa numeroa.

ChatGPT:llä meni noin 10–20 sekuntia vastauksen generointiin (kuvio 3), jonka se muotoilee kopioitavaksi ja vietäväksi testiympäristöön. ChatGPT generoi myös usein kommentit pyytämättä ohjelmakoodiin, jotta käyttäjä voi paremmin koodissa tapahtuu.



Kuvio 3. Testissä 1 tehtävän antaminen ChatGPT:lle

Alla (kuviossa 4) on ChatGPT:n generoima ohjelmakoodi kokonaisuudessaan sekä terminäli. ChatGPT:n generoima vastaus toimii ajettaessa oletetusti ja pyysi käyttäjää arvaamaan numeron minkä jälkeen ilmoitti, jos arvaus oli liian korkea vai liian matala. Ohjelmakoodi suoritetaan "def guessing\_game()" -funktion sisällä, jossa ohjelma valitsee satunnaisen muuttujan "random\_number" lukujen 0–100 väliltä.

```

1  import random
2
3  def guessing_game():
4      # Select a random number between 0 and 100
5      random_number = random.randint(0, 100)
6
7      # Initialize attempt count
8      attempts = 0
9
10     print("Welcome to the Guessing Game!")
11     print("I have selected a number between 0 and 100.")
12     print("Try to guess the number!")
13
14     while True:
15         # Ask the player to enter a guess
16         try:
17             guess = int(input("Enter your guess: "))
18         except ValueError:
19             print("Please enter a valid integer.")
20             continue
21
22         # Increment the attempt count
23         attempts += 1
24
25         # Check if the guessed number is correct, too high, or too low
26         if guess == random_number:
27             print(f"Congratulations! You've guessed the correct number {random_number}.")
28             print(f"It took you {attempts} attempts to guess the correct number.")
29             break
30         elif guess < random_number:
31             print("Your guess is too low. Try again!")
32         else:
33             print("Your guess is too high. Try again!")
34
35     # Run the guessing game
36     guessing_game()
37

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS Python Debug

```

Enter your guess: 70
Your guess is too low. Try again!
Enter your guess: 80
Your guess is too high. Try again!
Enter your guess: 77
Your guess is too high. Try again!
Enter your guess: 72
Your guess is too low. Try again!
Enter your guess: 75
Congratulations! You've guessed the correct number 75.
It took you 10 attempts to guess the correct number.

```

Kuvio 4. Testissä 1 ChatGPT:n generoima vastaus

Pelin toistorakenne toteutettiin "while True":lla, jossa pelaajaa pyydettiin arvaamaan satunnaisesti valittua numeroa, kunnes oikea numero on arvattu ja

ohjelma palauttaa onnitteluviestin pelin voittamisesta sekä arvauskertojen määrän. Generoitu vastaus on toimiva ja ohjelmakoodia on helppo ymmärtää ChatGPT:n lisäämien kommenttien avulla. Ohjelman toistorakenteen alussa tarkistettiin, että pelaaja antoi ohjelmalle kelpaavan numeron. Jos ohjelmakoodissa ei ole ehtorakenteita numeroiden tarkistamiseen niitä pyydettyäessä, niin usein se johtaa ohjelman vikatilaan.

### 3.1.2 Testi 2: Virheiden korjaus annetusta koodista

Testissä 2 tehtävänantona oli korjata annetusta ohjelmakoodista mahdolliset virheet. Ohjelmakoodi oli seuraavanlainen:


```
-----  
  
def sum_of_squares(numbers):  
  
    total = 0  
  
    for i in range(len(numbers)):  
  
        total += numbers[i] ** 2  
  
    return total  
  
numbers = [1, 2, 3, 4, 5]  
  
result = sum_of_squares(numbers)  
  
print("Sum of squares:", result)  
  
-----
```

Tehtävän 2 korjattava ohjelmakoodi syötettiin ChatGPT:lle (kuvio 5) verkkosivun käyttöliittymän kautta ja tarkennettiin pyynnössä, että kyseessä on virheellinen Python-ohjelmakoodi.



what is wrong with this python code: `def sum_of_squares(numbers):  
total = 0  
for i in range(len(numbers)):  
total += numbers[i] ** 2  
return total`

`numbers = [1, 2, 3, 4, 5]  
result = sum_of_squares(numbers)  
print("Sum of squares:", result)`

 The Python code you provided is logically correct and should work as intended to calculate the sum of squares of the numbers in the given list. However, there are a few stylistic improvements that can make the code more Pythonic:

1. **Using direct iteration over the list instead of `range(len(numbers))`:** This is more idiomatic and readable in Python.
2. **Adding type hints:** These can help with readability and maintainability.

Here is the improved version of your code:

```
python Copy code
def sum_of_squares(numbers: list[int]) -> int:
    total = 0
    for number in numbers:
        total += number ** 2
    return total

numbers = [1, 2, 3, 4, 5]
result = sum_of_squares(numbers)
print("Sum of squares:", result)
```

Kuvio 5. Testissä 2 virheellisen ohjelmakoodin syöttäminen ChatGPT:lle

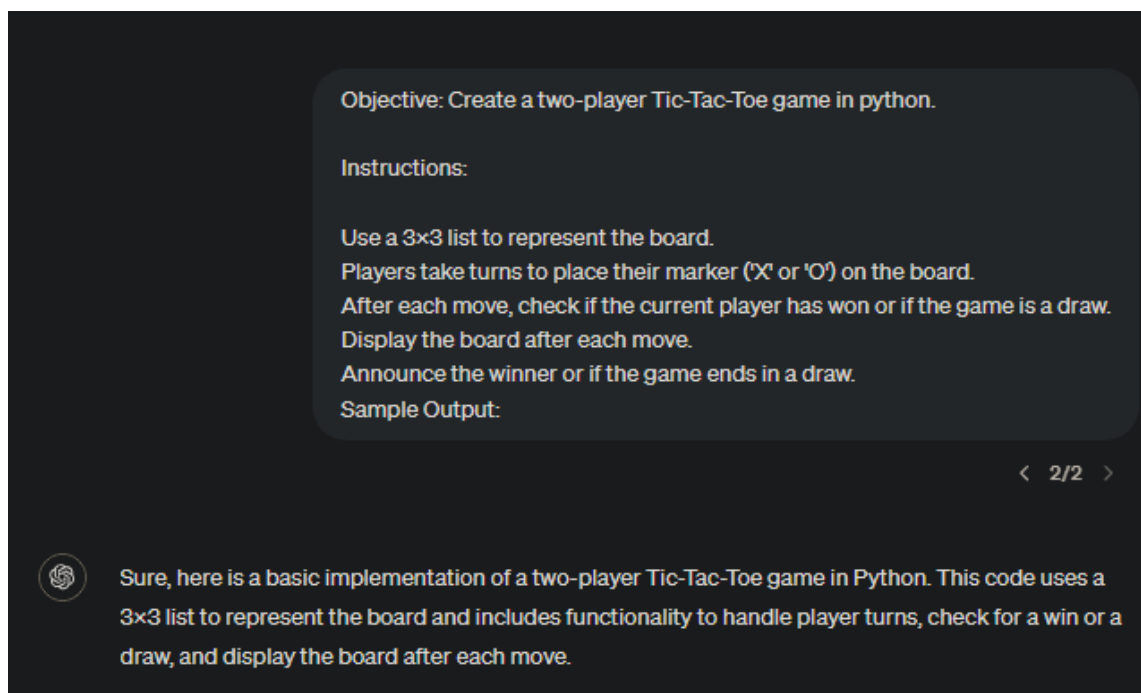
ChatGPT:n generoima vastaus korjaa toistorakenteen niin, että se iteroi jokaisen "numbers"-listan numeron yli. Alkuperäisessä ratkaisussa käytettiin "range(len(numbers))"-funktioita. ChatGPT lisäsi myös toistorakenteen alkuun tarkistuksen, jossa katsottiin ettei lista ollut tyhjä. ChatGPT:n generoima ohjelmakoodi korjaa alkuperäiset tehtävässä olleet virheet ja ohjelmakoodi palauttaa tulokseksi viisikymmentäviisi ajettaessa sen testiympäristössä.

### 3.1.3 Testi 3: Ristinollapeli

Tehtävässä 3 oli tehtävänantona luoda Python-kielellä ristinolla peli. Pelin tuli sisältää seuraavat ominaisuudet: luodaan "3x3" -lista edustamaan pelilautaa.

Pelaajat asettavat vuorotellen merkkinsä ('X' tai 'O') laudalle. Jokaisen siirron jälkeen tarkistettiin, oliko nykyinen pelaaja voittanut vai oliko peli tasapeli. Pelilauta näytettiin jokaisen liikkeen jälkeen, jossa voittaja ilmoitettiin pelaajille tai jos peli päättyi tasapeliin.

ChatGPT:llä meni noin 15–20 sekuntia generoida vastaus (kuvio 6) kokonaisuudessaan. Testissä 3 meni hieman kauemmin kuin aikaisimmilla testitapauksilla generoida vastaus, koska pyyntö on paljon vaikeampi ja vaatii useamman osan ohjelmakoodiin.



Kuvio 6. Testissä 3 pyynnön syöttäminen ChatGPT:lle

ChatGPT:n generoimassa vastauksessa oli alussa “def print\_board(board)” -funktio, joka ottaa yhden argumentin: “board”. Tämän odotetaan olevan kaksiulotteinen lista, jossa jokainen sisempi lista edustaa riviä pelilaudalla. Funktio (kuvio 7) tulostaa pelilaudan kutsuttaessa.

```

1  def print_board(board):
2      for row in board:
3          print(" | ".join(row))
4          print("-" * 5)
5

```

Kuvio 7. Testissä 3 ChatGPT:n generoima vastaus osa 1/5

"def check\_winner(board, player)" -funktio (kuvio 8) tarkistaa, onko kumpikaan pelaaja voittanut pelin. Se tekee tämän tarkistamalla kaikki mahdolliset voittoehdot ristinollapelissä: kolme peräkkäin vaaka, pysty tai vinottain peliruudulla.

```

6  def check_winner(board, player):
7      # Check rows
8      for row in board:
9          if all([cell == player for cell in row]):
10             return True
11
12     # Check columns
13     for col in range(3):
14         if all([board[row][col] == player for row in range(3)]):
15             return True
16
17     # Check diagonals
18     if all([board[i][i] == player for i in range(3)]) \
19         or all([board[i][2 - i] == player for i in range(3)]):
20         return True
21
22     return False
23
24 def check_draw(board):
25     return all([cell != ' ' for row in board for cell in row])
26

```

Kuvio 8. Testissä 3 ChatGPT:n generoima vastaus osat 2–3/4

Lisäksi "def check\_draw(board)" -funktio (kuvio 8) tarkistaa, onko peli tasapeli. Se tekee tämän tarkistamalla, olivatko taulukon kaikki solut täytetty (eli eivät ole yhtä suuria kuin ' '). Jos kaikki solut on täytetty, eikä voittajaa ole löydetty (joka olisi lopettanut pelin ennen kuin tämä toiminto kutsuttiin), peli on tasapeli. Funktio palauttaa arvon "True", jos peli on tasapeli, ja "False" muussa tapauksessa.

"def main()" -funktio (kuvio 9) on pelin aloitus. Se alustaa ensin tyhjän "3x3" -taulukon ja jokainen solu sisältää aluksi välilyönnin (' '). "current\_player"-muuttuja on asetettu arvoon "X", mikä tarkoittaa, että pelaaja "X" tekee ensimmäisen liikkeen pelissä.

```

27 def main():
28     board = [[' ' for _ in range(3)] for _ in range(3)]
29     current_player = 'X'
30
31     while True:
32         print_board(board)
33         print(f"Player {current_player}'s turn")
34
35         while True:
36             try:
37                 row = int(input("Enter the row (0, 1, 2): "))
38                 col = int(input("Enter the column (0, 1, 2): "))
39                 if board[row][col] == ' ':
40                     board[row][col] = current_player
41                     break
42                 else:
43                     print("The cell is already occupied. Try again.")
44             except (IndexError, ValueError):
45                 print("Invalid input. Please enter numbers between 0 and 2.")
46
47             if check_winner(board, current_player):
48                 print_board(board)
49                 print(f"Player {current_player} wins!")
50                 break
51
52             if check_draw(board):
53                 print_board(board)
54                 print("The game is a draw!")
55                 break
56
57             current_player = 'O' if current_player == 'X' else 'X'
58
59 if __name__ == "__main__":
60     main()
61

```

Kuvio 9. Testissä 3 ChatGPT:n generoima vastaus osa 4/4

Verrattuna aikaisempiin ChatGPT:n vastauksiin kommentointeja on huomattavasti vähemmän, mutta ohjelmakoodin toiminta oli selvä funktioiden osalta. Generoitu kokonaisuus oli toimiva ja

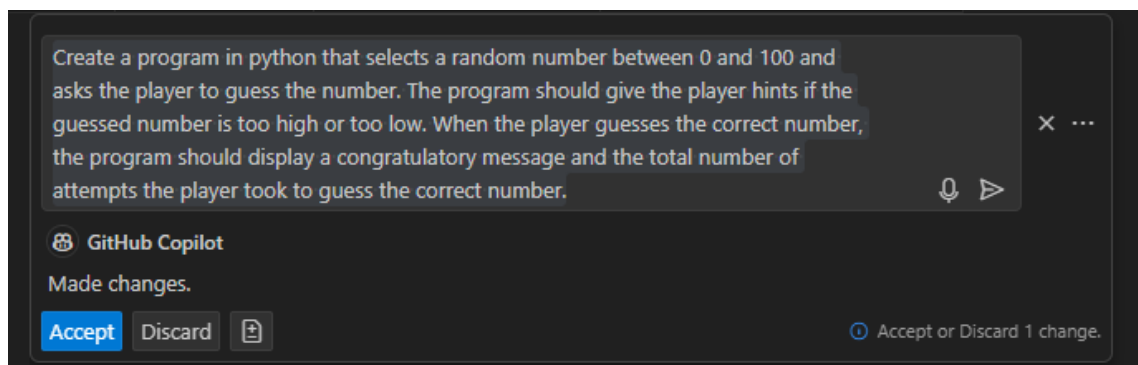
### 3.2 Copilot-ohjelmointitehtävät

Github Copilot -ohjelmointitehtävät suoritettiin ohjelmointiympäristön Visual Studio Code:lla mihin oli asennettu Copilot CLI, jonka avulla voidaan suoraan koodiriviltä tehdä AI:lle pyyntöjä. AI:n generoimat ratkaisut tehtäviin testattiin samalla tavalla kuin ChatGPT Visual Studio Coden sisällä Python-tiedostona.

### 3.2.1 Testi 1: Numeropeli

Testissä 1 tehtävänantona oli luoda ohjelma Python-kielellä, joka valitsee satunnaisluvun väliltä 0–100 ja pyytää pelaajaa arvaamaan luvun. Ohjelman tulee antaa pelaajalle vihjeitä, jos arvattu luku on liian suuri tai liian pieni. Kun pelaaja arvaa oikean luvun, ohjelman tulee näyttää onnitteluviesti. Pelin lopussa pelaajalle tulee näyttää, kuinka monta kertaa pelaaja on yrittänyt arvata oikeaa numeroa.

Copilotilla vastauksen generointiin meni noin 10–20 sekuntia, joka vastaa ChatGPT:n aikaa samassa tehtävässä. Copilot ei lisännyt kommentointeja vastaukseensa, joiden avulla selitettäisiin ohjelmakoodin toimintaa käyttäjälle.



Kuvio 10. Testin 1 tehtävän antaminen Copilotille

Tehtävän yksinkertaisuuden vuoksi Copilotilla ei ollut ongelmia vastauksen (kuvio 11) generoinnin kanssa ja ohjelma toimi, kuin tehtävänannossa pyydettiin. Ohjelma valitsi numeron ja pyysi pelaajaa arvaamaan numeroa. Ohjelma kertoi pelaajalle, jos arvaus on liian korkea tai liian matala sekä lopuksi ohjelma antoi onnitteluviestin oikeasta arvauksesta ja arvauksien määrän.

```
1 import random
2
3 def guess_number():
4     number = random.randint(0, 100)
5     attempts = 0
6
7     while True:
8         guess = int(input("Guess a number between 0 and 100: "))
9         attempts += 1
10
11         if guess < number:
12             print("Too low!")
13         elif guess > number:
14             print("Too high!")
15         else:
16             print(f"Congratulations! You guessed the number in {attempts} attempts.")
17             break
18
19 guess_number()
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS Python D

```
Too high!
Guess a number between 0 and 100: 76
Too high!
Guess a number between 0 and 100: 75
Too high!
Guess a number between 0 and 100: 74
Too high!
Guess a number between 0 and 100: 72
Too high!
Guess a number between 0 and 100: 71
Congratulations! You guessed the number in 11 attempts.
```

Kuvio 11. Testissä 1 Copilotin antama vastaus ja sen testaus terminaalissa

### 3.2.2 Testi 2: Virheiden korjaus annetusta koodista

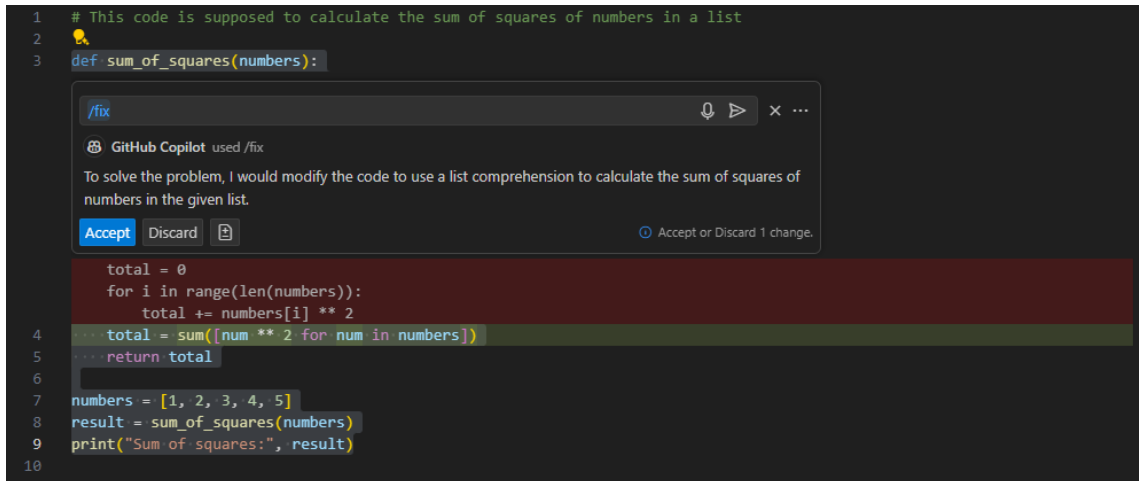
Testissä 2 mitattiin Copilotin kykyä löytää virheitä sille annetusta koodipätkästä ja sen korjausehdotuksien laatua. Testi 2 suoritettiin käyttämällä Copilotin yhtä CLI-toimintoa `/fix`, joka kohdistettiin valittuun ohjelmakoodiin ja generoi ehdotuksen ongelmien korjaamiseksi.

Testissä 2 tehtävänantona oli korjata annetusta ohjelmakoodista mahdolliset virheet. Ohjelmakoodi oli seuraavanlainen:

```
-----  
  
def sum_of_squares(numbers):  
  
    total = 0  
  
    for i in range(len(numbers)):  
  
        total += numbers[i] ** 2  
  
    return total  
  
numbers = [1, 2, 3, 4, 5]  
  
result = sum_of_squares(numbers)  
  
print("Sum of squares:", result)
```

```
-----
```

Copilotilla meni noin kaksikymmentä sekuntia generoida vastaus. Copilotin `"/fix"`-toiminto muutti (kuviossa 12) ohjelmakoodia siten, että `"range(len(numbers))"` sijaan käytettiin listaa numeroiden summan laskuun. Copilotin ehdottamat korjaukset kääntyi Visual Studio Code:ssa oikein ja ohjelma antoi oikean tuloksen, joka oli viisikymmentäviisi.



```

1 # This code is supposed to calculate the sum of squares of numbers in a list
2
3 def sum_of_squares(numbers):
4     total = 0
5     for i in range(len(numbers)):
6         total += numbers[i] ** 2
7
8     total = sum([num ** 2 for num in numbers])
9     return total
10
11 numbers = [1, 2, 3, 4, 5]
12 result = sum_of_squares(numbers)
13 print("Sum of squares:", result)

```

GitHub Copilot used /fix  
 To solve the problem, I would modify the code to use a list comprehension to calculate the sum of squares of numbers in the given list.

Accept or Discard 1 change.

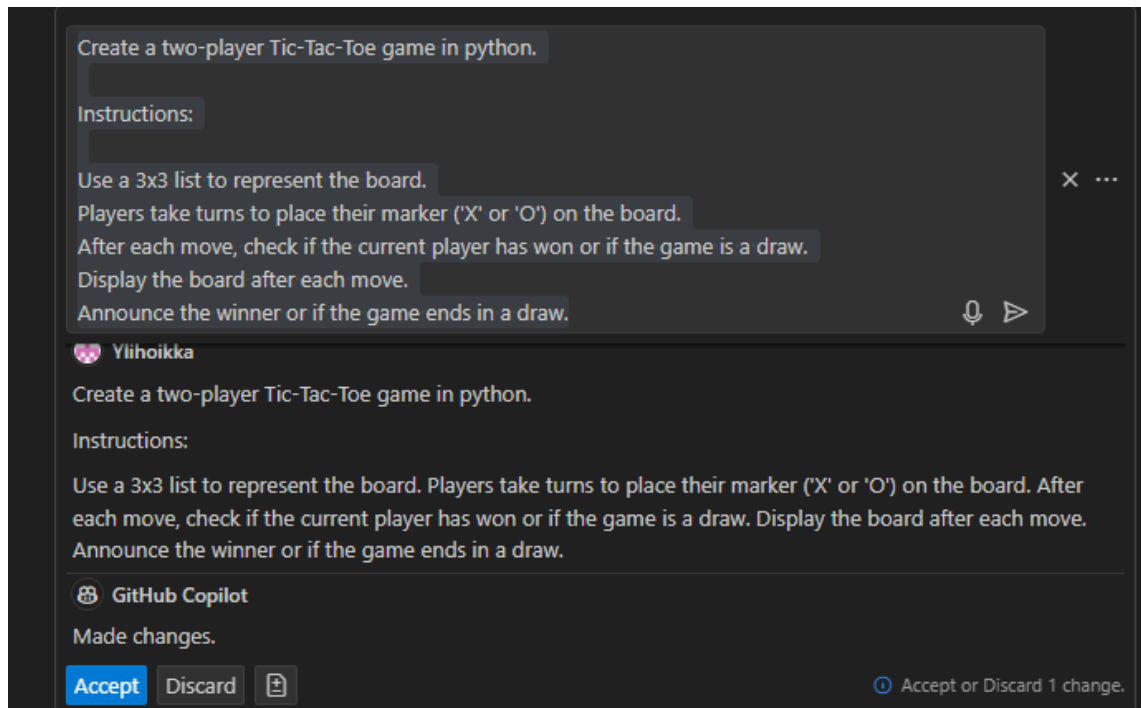
Kuvio 12. Testissä 2 Copilotin generoima vastaus

### 3.2.3 Testi 3: Ristinollapeli

Tehtävässä 3 oli tehtävänantona luoda Python-kielellä ristinolla peli. Pelin tuli sisältää seuraavat ominaisuudet: luodaan "3x3" -lista edustamaan pelilautaa. Pelaajat asettavat vuorotellen merkkinsä ('X' tai 'O') laudalle. Jokaisen siirron jälkeen tarkistettiin, oliko nykyinen pelaaja voittanut vai oliko peli tasapeli. Pelilauta näytettiin jokaisen liikkeen jälkeen, jossa voittaja ilmoitettiin pelaajille tai jos peli päättyy tasapeliin.

Kuviossa 13 näkyy tehtävänannon syöttäminen Visual Studio Coden sisällä, jossa Copilotilla meni noin 10–15 sekuntia generoida vastaus pyyntöön. Pyyntö syötettiin Copilotin CLI:n kautta generoitavaksi englannin kielellä.





Kuvio 13. Testissä 3 pyynnön syöttäminen Copilotille

Ohjelmakoodissa "def create\_board()" -funktio (kuvio 14) alusti "3x3" -taulukon, joka edustaa ristinollapeliä. Tämä tehtiin "list comprehension" avulla, joka on tapa luoda luetteloita Python-kielellä.

"def display\_board(board)" -funktio (kuvio 14) otti pelilaudan "board" argumentiksi, ja iteroi jokaisen taulukon rivin for-silmukalla. Minkä jälkeen tulosti sen terminaaliin pelaajan nähtäväksi.

```

1  def create_board():
2      board = [[' ' for _ in range(3)] for _ in range(3)]
3      return board
4
5  def display_board(board):
6      for row in board:
7          print(' '.join(row))
8          print('-' * 5)
9

```

Kuvio 14. Testissä 3 Copilotin generoima vastaus osat 1–2/5

"def is\_winner(board, player)" -funktio (kuvio 15) toistui jokaisen taulukon rivin kohdalla. Jokaisella rivillä se tarkisti, sisältävätkö rivin solut pelaajan merkin X- tai O:n.

```

9
10 def is_winner(board, player):
11     # Check rows
12     for row in board:
13         if all(cell == player for cell in row):
14             return True
15
16     # Check columns
17     for col in range(3):
18         if all(board[row][col] == player for row in range(3)):
19             return True
20
21     # Check diagonals
22     if all(board[i][i] == player for i in range(3)):
23         return True
24     if all(board[i][2-i] == player for i in range(3)):
25         return True
26
27     return False

```

Kuvio 15. Testissä 3 Copilotin generoima vastaus osa 3/5

"def is\_draw(board)" -funktio (kuvio 16) tarkisti tasapelin etsimällä taulukolta tyhjiä soluja. Se teki tämän iteroimalla jokaisen taulukon rivin.

```

28
29 def is_draw(board):
30     for row in board:
31         if ' ' in row:
32             return False
33     return True

```

Kuvio 16. Testissä 3 Copilotin generoima vastaus osa 4/5

Peliä ohjasi "def play\_game()" -funktio (kuvio 17), joka loi ensin tyhjän pelilaudan ja asetti nykyisen pelaajan muuttujan arvoon "X". Sitten ohjelma siirtyi while-toistorakenteeseen, jossa ohjelmakoodi tulosti ruudut näkyville ja ilmoitti kenen pelaajan vuoro oli seuraavaksi terminaaliin. Vuoron jälkeen peli pyysi pelaajalta mihin ruutuun pelaaja haluaisi laittaa merkkinsä, ja tarkisti ettei siinä ruudussa ollut valmiiksi merkkiä.

```

34
35 def play_game():
36     board = create_board()
37     current_player = 'X'
38
39     while True:
40         display_board(board)
41         print(f"Player {current_player}'s turn.")
42
43         row = int(input("Enter the row (0-2): "))
44         col = int(input("Enter the column (0-2): "))
45
46         if board[row][col] != ' ':
47             print("Invalid move. Try again.")
48             continue
49
50         board[row][col] = current_player
51
52         if is_winner(board, current_player):
53             display_board(board)
54             print(f"Player {current_player} wins!")
55             break
56
57         if is_draw(board):
58             display_board(board)
59             print("It's a draw!")
60             break
61
62         current_player = 'O' if current_player == 'X' else 'X'
63
64     play_game()

```

Kuvio 17. Testissä 3 Copilotin generoima vastaus osa 5/5

Ohjelmakoodi tarkisti jokaisen vuoron jälkeen, jos millään rivillä oli kolmen suoraa samaa merkkiä. Peli jatkui niin kauan, kunnes toinen pelaaja sai kolmen suoran tai peli päättyi tasapeliin. Alla on kuvio 18 Visual Studio Coden terminaalista, jossa näkyy pelin lopetus.

```

Player X's turn
Enter the row (0, 1, 2): 2
Enter the column (0, 1, 2): 2
X | O |
-----
  | X |
-----
O |  | X
-----
Player X wins!

```

Kuvio 18. Testin 3 Copilotin ristinolla-voitto

### 3.3 Tuloksien vertailu ja analyysi

Testissä 1 Copilotin generoimassa vastauksessa käytettiin samaa metodia funktion sisällä kuin ChatGPT:n generoimassa vastauksessa. "While True"-toistorakenteen alussa pyydettiin käyttäjältä numeroa, mutta sitä ei tarkistettu Copilotin vastauksesta, ellei annettu arvo ollut integer-tyyppinen. Mikäli käyttäjä antoi muun kuin integer-arvon niin ohjelmakoodi lopetti toiminnan. Mutta ChatGPT:n generoimassa ratkaisussa se tarkistettiin ja ohjelma ei edennyt, jos arvo oli muu kuin integer.

Ohjelman while-toistorakenteessa ohjelmakoodi vertasi annettua arvoa valittuun numeroon, kunnes arvaus oli oikein ja ohjelma antoi onnitteluviestin ja arvauskertojen määrän. Copilot-ohjelmakoodi vertasi "else if" -rakenteella arvaus-muuttujaa, mikäli se oli suurempi tai pienempi kuin valittu luku.

ChatGPT:n vastauksessa arvausta verrattiin ensin suoraan valittuun oikeaan lukuun, jonka jälkeen "else if" -rakenteella, onko se suurempi tai pienempi kuin valittu luku. Periaate oli molemmissa ohjelmakoodeissa sama, mutta toteutus on hieman eri molemmissa. Copilotin antama vastaus oli selkeämmin toteutettu kuin ChatGPT:n, mutta ChatGPT lisäsi annetun muuttujan tarkistamisen ohjelmakoodiin, joka olisi hyvä olla molemmissa. Molemmat suoriutuivat kiitettävästi annetusta tehtävästä.

Testissä 2 ChatGPT ja Copilot löysivät molemmat virheet ohjelmakoodista sekä korjasivat ne hieman eri tavoilla. ChatGPT:n vastauksessa toistorakenne kävi suoraan "numbers" -listan läpi iteroimalla ja tarkisti, ettei lista ollut tyhjä. Copilotin generoimassa ratkaisussa luotiin uusi lista arvoista "numbers" -listassa ja laskettiin niiden summa. ChatGPT:n lisäämä tarkistus listan arvoille oli toki hyvä olla, jos ohjelmakoodi olisi osana jotain suurempaa kokonaisuutta. Molempien tekoälyjen generoima vastaus täytti tehtävänannon halutut ominaisuudet ja lopputulos oli kiitettävä.

Testissä 3 Copilot sekä ChatGPT onnistuivat generoimaan nopeasti ymmärrettävän ohjelmakoodin tehtävänannon mukaisesti, mutta ohjelmakoodin selvyuden kannalta Copilotin generoima vastaus oli mielestäni selvempi. Ratkaisuissa oli pieniä eroavaisuuksia, mutta toiminta oli molemmissa miltei

identtinen ja molemmat täytti tehtävänannon vaatimukset. Funktio millä pelilauta alustettiin puuttui ChatGPT:n vastauksesta kokonaan, koska se oli laitettu sisällytetty "def main()"- pääfunktion sisälle.

Ohjelmien toistorakenteet erosivat toisistaan siinä, että ChatGPT:n ratkaisussa oli enemmän ominaisuuksia verrattuna Copilotin ratkaisuun. ChatGPT:n "while" -toistorakenteessa eriteltiin, mikäli pelaaja yritti laittaa jo täytettyyn ruutuun merkkiä ja jos annettu ruutunumero oli muu kuin 0–2 -arvot. Molempien tekoälyjen generoima vastaus täytti tehtävänannon halutut ominaisuudet. Tekoälyt kykenivät suoriutumaan kiitettävästi niille annetuista ohjelmointitehtävistä ja generoidut vastaukset olivat myös toimivia, joka vastasi tutkimuskysymykseen "kuinka hyvin tekoälyt ChatGPT ja Copilot kykenevät suoriutumaan ohjelmointipainotteisista-tehtävistä?".

Tutkimusosioiden tuloksista pystyi päätellä, että vaikka tekoälyjen generoimat vastaukset ovat laadukkaita, niiden generoimat vastaukset/käytetyt metodit voivat vaihdella paljonkin. Missä vaiheessa tekoälyt ovat ja tarviiko kehittäjien muokata tekoälyjen generoimia vastauksia? -tutkimuskysymykseen saatiin vastaus tarkastelemalla tekoälyjen generoimia tutkimusosion kokonaisuuksia. Tekoälyt ovat siinä vaiheessa, että pienet ja keskisuuret kokonaisuudet onnistuvat helposti, kuten tutkimusosiossa todettiin ohjelmointitehtävillä. Kaikki tekoälyjen tuottama materiaali on käytävä tarkasti läpi, koska tekoälyt ovat myös alttiita virheille.

ChatGPT:n ja Copilotin käyttöönotto-prosessi oli tehty todella helpoksi ja suoraviivaiseksi. Näiden pohjalta saatiin vastaus tutkimuskysymykseen "kuinka helppoa on integroida tekoälyt osaksi ohjelmoinnin työkaluja?". ChatGPT vaatii vain kirjautumisen sivulle, ja tekoäly oli valmis käytettäväksi verkkosivun käyttöliittymästä. Copilot vaatii enemmän askelia käyttöönotossa, kuten maksutavan lisääminen ja asennus haluamaan ohjelmointiympäristöön, joka oli tässä tapauksessa Visual Studio Code.

ChatGPT lisäsi usein ylimääräisiä osia ohjelmakoodiin, joita ei erikseen pyydetty. Osa ChatGPT:n lisäämistä toiminnallisuuksista oli hyviä lisäyksiä ohjelmakoodin toiminnallisuuden kannalta, kuten ohjelmakoodin kommentointi.

Copilot antoi tutkimustehtävien perusteella aina toimivan ratkaisun, jossa on vain osat mitä tarvitaan ohjelman suorittamiseen pyydetyllä tavalla. Copilotin antamia ratkaisuja oli helppo lukea, koska niissä ei ollut mitään ylimääräistä mitä ei pyydetty.

Käytettävyydeltään Copilot soveltuu loistavasti ohjelmointiin aloittelijoille ja kokeineille kehittäjille. Copilot sisältää myös paljon ominaisuuksia, joilla voidaan nopeuttaa työskentelyä. Ohjelmointiympäristöjen sisään integroidut tekoälytyökalut tulevat yleistymään aja kuluessa, koska verrattuna ChatGPT:n verkkosivu-käyttöliittymään Copilot oli huomattavasti nopeampaa käyttää ohjelmoinnissa ja sen generoimat vastaukset olivat täsmällisempiä. ChatGPT:lle oli myös saatavilla laajennuksia ohjelmistokehitysympäristöihin.

#### 4 POHDINTA

En ollut aikaisemmin käyttänyt tekoälyjä ohjelmoinnin työkaluina, joten tämän opinnäytetyön työstäminen oli itselle myös todella opettava prosessi. Varsinkin Copilotin avulla vanhat asiat ohjelmoinnista palasivat nopeammin mieleen, eikä ajatteluun ja tiedonhakuun mennyt läheskään niin paljon aikaa. Ohjelmointiympäristöön integroidut tekoälyt, kuten Copilot CLI on hyvä työkalu ohjelmakoodin ymmärtämiseen ja opiskeluun.

Kaikki tekoälyn generoima materiaali ohjelmointimaailmassa vaatii ihmisen suorittamaa tarkastelua ja auditointia, ennen kuin sitä voidaan käyttää missään mittakaavassa luotettavasti. Vaikka suurin osa tekoälyjen generoimista ratkaisuista toimivat, voi joukossa silti olla virheitä. Pienet virheet eivät nouse esille kokemattomaan silmään ja ovat hyvien ohjelmointikäytäntöjen vastaisia.

Tekoälyjen käyttö on hyvä kohdistaa ohjelmoinnissa pienistä keskisuuriin skripteihin ja funktioihin, joissa virheet on nopea tunnistaa ja generoidut ratkaisut pitivät usein paikkansa. Tällöin työtuottavuus nousee, mutta aikaa ei kulu liikaa tekoälyn mahdollisten virheiden metsästykseseen. Ohjelmoinnin apuna tekoälyjä voidaan käyttää myös suunnittelutyössä, jossa kokonaisuuksien hahmottelussa voi olla hyötyä. Toistuvien tehtävien suorittaminen, kuten ohjelmakoodin debugaaminen voidaan siirtää tekoälyn mietittäväksi. Copilot ja ChatGPT voidaan kouluttaa keskustelemalla generoimaan esimerkiksi dokumentaatio kehittäjän haluamalla tavalla.

Lähdetietoja ChatGPT:n ilmainen GPT-3.5 -kielimalli ei kykene kertomaan tarkasti, mihin generoidun tekstin tieto perustuu. Vastaus on usein ympärilyöreä, koska usein generoitu vastaus voi sisältää dataa lukuisista eri lähteistä.

ChatGPT ja Copilot tarjoavat kehittäjille älykästä apua koodin luomisessa, virheiden tunnistamisessa ja suorituskyvyn optimoinnissa. Nämä työkalut voivat automatisoida rutiinitehtävät, jolloin kehittäjät voivat keskittyä enemmän ohjelmistosuunnitteluun ja -kehitykseen. Lisäksi ne voivat toimia arvokkaina resursseina vähemmän kokeneille ohjelmoijille tarjoten ohjausta ja tukea reaaliajassa. Ohjelmoinnin opiskelijat voisi hyötyä tuutorin tapaisesta tekoälystä,

joka ei suoraan kertoisi vastausta oppilaalle vaan neuvoisi ja antaisi oppilaan itse miettiä ja rakentaa itse ohjelmistokoodia.

Ohjelmoinnista kiinnostuneille suosittelen kokeilemaan tekoälyä työkaluina itseoppimiseen ja mielenkiinnon herättämiseen ohjelmointiin. On hyvä kuitenkin muistaa, että tekoäly on kuitenkin vain työkalu, joka tukee ja vahvistaa olemassa olevaa osaamista.



## LÄHTEET

Domhke, T 2023. Universe 2023: Copilot transforms GitHub into the AI-powered developer platform, Github. Viitattu 16.5.2024. nettiosoite.

<https://github.blog/2023-11-08-universe-2023-copilot-transforms-github-into-the-ai-powered-developer-platform/>

Github 2022a. Features. Viitattu 15.4.2024 nettiosoite.

<https://github.com/features/copilot>

– Github 2022b. Getting started with GitHub Copilot. Viitattu 15.4.2024 nettiosoite. <https://github.com/features/copilot/getting-started>

Github 2023. About GitHub Copilot in the CLI. Viitattu 15.4.2024 nettiosoite.

<https://docs.github.com/en/copilot/github-copilot-in-the-cli/about-github-copilot-in-the-cli>

IBM 2023. What is AI. Viitattu 8.5.2024 nettiosoite.

<https://www.ibm.com/topics/artificial-intelligence>

Kallioniemi, I. 2022. Tekoälyoikeus: Varallisuus oikeuden ja riskienhallinnan kysymyksiä. Helsinki: Alma Talent. Viitattu 28.5.2024 nettiosoite.

[https://verkkokirjahylly-almatalent-fi.ez.lapinamk.fi/teos/BAHBFXETEB#kohta:5\(\(20\)\)\(7c\)\(\(20\)TEKO\(\(c4\)LYJ\(\(c4\)R JESTELM\(\(c4\)N\(\(20\)OPETUSAINEISTON\(\(20\)SUOJA\(:5.4\(\(20\)Tietokantojen\(\(20\)rakenteen\(\(20\)suoja\(:5.4.2\(\(20\)Kirjallisesti\(\(20\)kuvattu\(\(20\)rakenne/piste:t4e](https://verkkokirjahylly-almatalent-fi.ez.lapinamk.fi/teos/BAHBFXETEB#kohta:5((20))(7c)((20)TEKO((c4)LYJ((c4)R JESTELM((c4)N((20)OPETUSAINEISTON((20)SUOJA(:5.4((20)Tietokantojen((20)rakenteen((20)suoja(:5.4.2((20)Kirjallisesti((20)kuvattu((20)rakenne/piste:t4e)

Kananen, H., & Puolitaival, H. 2019. Tekoäly: Bisneksen uudet työkalut. Alma Talent Oy. Viitattu 19.5.2024 nettiosoite. [https://bisneskirjasto-almatalent-fi.ez.lapinamk.fi/teos/BAXBBXATCBIED#kohta:OSA\(\(20\)1\(\(20\)Teko\(\(e4\)lyn\(\(20\)mahdollisuudet/piste:tEd](https://bisneskirjasto-almatalent-fi.ez.lapinamk.fi/teos/BAXBBXATCBIED#kohta:OSA((20)1((20)Teko((e4)lyn((20)mahdollisuudet/piste:tEd)

Kananen, H., & Puolitaival, H. 2019. Tekoäly: Bisneksen uudet työkalut. Alma Talent Oy. Viitattu 19.5.2024 nettiosoite. [https://bisneskirjasto-almatalent-fi.ez.lapinamk.fi/teos/BAXBBXATCBIED#kohta:OSA\(\(20\)1\(\(20\)Teko\(\(e4\)lyn\(\(20\)mahdollisuudet/piste:tEd](https://bisneskirjasto-almatalent-fi.ez.lapinamk.fi/teos/BAXBBXATCBIED#kohta:OSA((20)1((20)Teko((e4)lyn((20)mahdollisuudet/piste:tEd)

Microsoft. 2023. Visual Studio Code Viitattu 30.5.2024 nettiosoite.

<https://code.visualstudio.com/docs/copilot/prompt-crafting>

Oguz, A. 2023. AI Prompt Engineering Isn't the Future. Harvard Business Review. Viitattu 28.5.2024 nettiosoite. <https://hbr.org/2023/06/ai-prompt-engineering-isnt-the-future>

Oguz, A. 2023. AI Prompt Engineering Isn't the Future. Harvard Business Review. Viitattu 28.5.2024 nettiosoite. <https://hbr.org/2023/06/ai-prompt-engineering-isnt-the-future>

Ojanperä, T. 2023. Tekoälyn vallankumous: Käsikirja. Helsinki: Alma Talent.

Viitattu 15.4.2023 nettiosoite. [https://bisneskirjasto-almatalent-fi.ez.lapinamk.fi/teos/CAHBBXXTBBAEF#kohta:Teko\(\(e4\)lyn\(\(20\)vallankumous/piste:thl](https://bisneskirjasto-almatalent-fi.ez.lapinamk.fi/teos/CAHBBXXTBBAEF#kohta:Teko((e4)lyn((20)vallankumous/piste:thl)

OpenAI 2022a. About. Viitattu 16.5.2024 nettiosoite. <https://openai.com/about>

– OpenAI 2022b. Introducing ChatGPT. Viitattu 16.5.2024 nettiosoite.

<https://openai.com/index/chatgpt/>

– OpenAI 2022c. Pricing. Viitattu 16.5.2024 nettiosoite.

<https://openai.com/chatgpt/pricing/>

Salo, I. 2023. Luova tekoäly mullistaa kaiken: ChatGPT näyttää tietä. Helsinki: Kauppakamari. Viitattu 8.5.2023 nettiosoite. [https://kauppakamaritieto.fi.ez.lapinamk.fi/ammattikirjasto/teos/luova-tekoaly-muuttaa-kaiken-2023#kohta:1.\(\(20\)Teko\(\(e4\)lyn\(\(20\)historia\(\(20\)lyhyesti\(:1.1\(\(20\)Teko\(\(e4\)lyn\(\(20\)varhaiset\(\(20\)vaiheet](https://kauppakamaritieto.fi.ez.lapinamk.fi/ammattikirjasto/teos/luova-tekoaly-muuttaa-kaiken-2023#kohta:1.((20)Teko((e4)lyn((20)historia((20)lyhyesti(:1.1((20)Teko((e4)lyn((20)varhaiset((20)vaiheet)

Shashkevich, A. 2019. Stanford researcher examines earliest concepts of artificial intelligence, robots in ancient myths. Stanford report. Viitattu 26.5.2024 nettiosoite. <https://news.stanford.edu/stories/2019/02/ancient-myths-reveal-early-fantasies-artificial-life>

Shaw, J. 2023. Embracing AI, The dawn of the virtual teaching fellow. Harvard Magazine. Viitattu 30.5.2024 nettiosoite. <https://www.harvardmagazine.com/2023/08/ai-in-education>