

Opinnäytetyö (AMK)

Tietojenkäsittely

2023

Joose Kirmonen

# OpenAI GPT-3.5 -tekoälyn käyttö ohjelmoinnin tukena



Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tietojenkäsittely

2023

Joose Kirmonen

## OpenAI GPT-3.5 -tekoälyn käyttö ohjelmoinnin tukena

Opinnäytetyön tavoitteena oli kartoittaa OpenAI:n kehittämän ChatGPT-työkalun version 3.5 käyttöä ohjelmoinnin tukena. Työssä selvitettiin testejä tekemällä, kuinka työkalu voi olla hyödyksi erilaisissa ohjelmoinnin osa-alueissa. Testausta varten luotiin kehikko, johon valittiin erilaisia tilanteita, joita ohjelmoija saattaa kohdata työssään päivittäin. Työssä käytiin myös läpi erilaisia työkaluun liittyviä tekoälyn käsitteitä ja sitä, kuinka ne liittyvät testattavaan työkaluun. Lopuksi kartoitettiin työkalun ja tekoälyn tulevaisuuden mahdollisuuksia erityisesti ohjelmoinnin näkökulmasta.

Testausvaiheessa käytiin läpi seuraavia asioita: koodin tuottaminen, vaihtoehtojen tuottaminen, esimerkkitiedon tuottaminen, datan muotoilu, koodin selittäminen, dokumentointi, virheiden tarkistus, virheiden korjaaminen, testien tuottaminen, koodin optimointi, koodin kääntö toiselle kielelle sekä tiedon etsiminen.

Työkalu suoriutui sille esitetystä vaatimuksista hyvin. Testausvaiheessa käytiin läpi edellä mainittuja ohjelmoinnin osa-alueita, ja työkalu oli niistä jokaisessa hyödyllinen apuri. Testauksen tulokset kertovat sen, että GPT-3.5:n kaltaiset työkalut tulevat tulevaisuudessa olemaan hyvin suuri osa ohjelmointia ja ohjelmoijan työnkuvaa.

Asiasanat:

Tekoäly, koneoppiminen, neuroverkot, älykkäät agentit, ohjelmointi, ohjelmointiympäristö.

Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Business Administration

2023

Joose Kirmonen

## Use of OpenAI GPT-3.5 artificial intelligence to support programming

The aim of the thesis was to map the use of version 3.5 of the ChatGPT tool developed by OpenAI as a programming support. Tests were carried out to find out how the tool can be useful in various areas of programming. For the testing, a framework was created, in which various situations that a programmer might encounter in his daily work were selected. The work also goes through various artificial intelligence concepts related to the tool and how they relate to the tool in question. In the end we looked at the future possibilities of the tool and artificial intelligence in general, especially from the perspective of programming.

In the testing phase, the following things were covered: generating code, generating alternatives, generating sample data, formatting data, explaining the code, documenting, checking for errors, correcting errors, generating tests, optimizing the code, translating the code into another language, and searching for information.

The tool performed well in terms of the requirements presented to it. During the testing phase, the aforementioned areas of programming were reviewed, and the tool was a useful helper in all of them. The results of the testing show that tools like GPT-3.5 will be a very large part of programming and a programmer's job description in the future.

Keywords:

Artificial intelligence, machine learning, neural networks, intelligent agents, programming, programming environment.

# Sisältö

<b>Lyhenteet</b>	<b>6</b>
<b>1 Johdanto</b>	<b>7</b>
<b>2 Tekoälyn käsitteitä</b>	<b>9</b>
2.1 Tekoäly	9
2.2 Koneoppiminen	9
2.3 Neuroverkot	10
2.4 Syväoppiminen	12
2.5 GPT ja kielimalli	13
<b>3 GPT-3.5:n testaus</b>	<b>15</b>
3.1 Koodin tuottaminen	16
3.2 Esimerkkidatan tuottaminen	21
3.3 Koodin selittäminen	23
3.4 Koodin dokumentointi	26
3.5 Virheiden tarkistus	28
3.6 Yksikkötestit	30
3.7 Koodin optimointi	32
3.8 Koodin käänntö toiselle kielelle	36
3.9 Tiedon etsiminen	37
<b>4 Tulevaisuuden näkymiä ja johtopäätöksiä</b>	<b>39</b>
<b>Lähteet</b>	<b>42</b>

## Kuvat

Kuva 1. Neuroverkon rakenne (IBM 2023c, muokattu).	11
Kuva 2. Kielimallien parametrien määrä miljardeissa.	14
Kuva 3. Työkalulle annettu syöte.	16
Kuva 4. Koodin tuottaminen.	17
Kuva 5. Tuotetun koodin selitys.	17
Kuva 6. Tuotetun koodin ajaminen.	17
Kuva 7. Satunnainen tuotettu koodi, selitys sekä tuloste.	19
Kuva 8. Vaihtoehtoinen ratkaisu.	20
Kuva 9. Tuotettu esimerkkidata.	21
Kuva 10. Muokattu esimerkkidata.	22
Kuva 11. Annettu syöte ja aremmin järjestetty esimerkkidata.	23
Kuva 12. Selitys koodin toiminnasta.	26
Kuva 13. Koodin dokumentointi.	27
Kuva 14. Virheellisen koodin korjaus.	30
Kuva 15. Yksikkötestien koodi ja selitys.	31
Kuva 16. Ohjeet numeropelin optimointiin.	34
Kuva 17. Osa optimoitua koodia.	35
Kuva 18. Toiselle kielelle käännetty koodi.	37
Kuva 19. Tiedon etsiminen.	38

## Lyhenteet

AGI	Artificial General Intelligence. Yleinen tekoäly on tekoälyn teoreettinen muoto, jota käytetään kuvaamaan tiettyä tekoälyn kehittämisen ajattelutapaa. Tavoitteena on luoda älykkäitä koneita, joita ei voi erottaa ihmismielestä. (IBM 2023.)
GPT	Generative Pre-trained Transformer. Generatiivinen esikoulutettu muuntaja, joka kuuluu neuroverkkojen perheeseen. Tätä muuntaja-arkkitehtuuria käytetään erityisesti generatiivisissa sovelluksissa, kuten ChatGPT. (OpenAI 2023.)

# 1 Johdanto

Tämä opinnäytetyö käsittelee OpenAI:n kehittämää ChatGPT-työkalua ja erityisesti sen 3.5-versiota. Työssä keskitytään pääosin siihen, miten työkalu voi nyky muodossaan auttaa ohjelmoinnin erilaisissa osa-alueissa ja toimia ohjelmoijan apuna. Keväällä 2023 3.5 versio on vielä ilmainen ja kaikkien vapaasti käytettävissä. OpenAI on myös julkaissut jo kehittyneemmän 4.0-version, joka on monin tavoin kehittyneempi kuin tämän työn aiheena oleva 3.5-versio. (OpenAI 2023.) Uudempi versio ei kuitenkaan ole vapaasti saatavilla, eikä odotuslistan pituudesta ole keväällä 2023 tarkempaa tietoa. Työkalun testaamista varten on koottu kehikko, jonka avulla käydään läpi erilaisia ohjelmoinnin osa-alueita ja testataan, kuinka hyvin GPT-3.5 osaa auttaa näiden kanssa. Varsinaiset testit tehdään Visual Studio Coden avulla.

Opinnäytetyön tavoitteena on kartoittaa, kuinka GPT-3.5:n kaltainen tekoäly voi jo nykyhetkellä olla avuksi ohjelmoinnissa ja minkälaista apua ohjelmoija voi työkalulta odottaa. Vaikka tekoälyn tehostamat työkalut tulevat vielä kehittymään valtavasti, on nyt hyvä hetki tutustua niiden käyttöön tulevaisuutta varten. Työssä tehtävät testit tehdään Python-ohjelmointikielellä, mutta testien tuloksia voidaan hyvin soveltaa myös muihin ohjelmointikieliin. Yksi suuri rajoite GPT-3.5-versiolla on sen koulutusdatan loppuminen vuoteen 2021, eli tämän jälkeisistä tapahtumista tai kehityksistä työkalun ilmainen versio ei tiedä mitään. (Brown ym. 2020.) Maksullisessa versiossa on mukana myös mahdollisuus antaa tekoälyn selata verkkosivuja, mikä antaa tekoälylle pääsyn päivitettyyn tietoon. Koska ohjelmointi on hyvin nopeasti muuttuva ja kehittyvä ala, tiedon rajaus vuoteen 2021 voi aiheuttaa ongelmia nykyaikaisten työkalujen käytössä.

Tekoäly tulee varmasti mullistamaan monta osa-aluetta nykyihmisen elämästä, ohjelmointi on vain pieni osa siitä, mihin nämä työkalut tulevat pian kykenemään. Tekoälyn kehitys on lähtenyt räjähdysmäiseen kasvuun viime vuosien aikana, vaikka monet näiden takana olevista ideoista ovat vuosikymmeniä vanhoja. Ensimmäiset ideat koneellisesta älykkyydestä juontavat juurensa 1940-luvulle ja vuonna 1956 pidettyä Darthmouthin kesän

tutkimusprojektia pidetään tekoälyn syntymäpaikkana (Darthmouth College 2023). Kehityksen kiihtymisen takana on useampi erillinen tekijä, jotka yhdessä ovat mahdollistaneet nykytilanteen. Tietokoneiden laskentateho ja kapasiteetti jatkavat kasvuaan, mikä vaikuttaa suoraan tekoälyn tehokkuuteen. Osaltaan kehitykseen vaikuttavat myös tiedon määrä ja sen saatavuus, nykyiset tekoälymenetelmät eivät olisi mahdollisia ilman sitä valtavaa tiedon määrää, joka on nykyään helposti saatavilla. Uudet tekoälymenetelmät, kuten neuroverkkoihin perustuva syväoppiminen on mahdollistanut monimutkaisempia tehtäviä, kuten esimerkiksi kuvien tunnistamisen. Suuri vaikutus kehitykseen on myös tekoälyn tehdyillä investoinneilla, jotka ovat kiihtyneet kehityksen mukana. Yritykset ovat huomanneet tekoälyn potentiaalin liiketoiminnan eri osa-alueilla. (OpenAI 2023.)

Opinnäytetyössä käsitellään ensin tekoälyn liittyviä käsitteitä ja kartoittaa hieman tekoälyn historiaa ja kehitysvaiheita. Tämän jälkeen keskitytään siihen, mitä kaikkea työkalulla pitäisi olla teoriassa mahdollista tehdä tällä hetkellä. Mahdollisuuksien kartoittamisen jälkeen testataan ominaisuuksia käytännössä käyttäen määriteltyä kehikkoa, näiden testien tuloksia esitellään kuvakaappauksilla ja koodin pätkillä. Lopuksi käydään vielä läpi tämänkaltaisten työkalujen tulevaisuuden näkymiä ja tehdään yhteenveto työkalun käyttömahdollisuuksista ohjelmoinnin tukena.



## 2 Tekoälyn käsitteitä

### 2.1 Tekoäly

Tekoälystä on tullut eräänlainen kaiken kattava termi sovelluksille tai koneille, jotka suorittavat monimutkaisia tehtäviä. Ennen monet näistä tehtävistä vaativat ihmisen väliintuloa toimiakseen, mutta tekoälyn tavoitteena on suorittaa näitä tehtäviä inhimillisten piirteiden avulla. Näihin lukeutuvat esimerkiksi taito järkeillä, tehdä yleistyksiä ja oppia vanhempien kokemusten perusteella. Aina digitaalisen tietokoneen kehityksestä asti 1940-luvulla on pystytty osoittamaan, että koneet voidaan ohjelmoida suorittamaan hyvinkin monimutkaisia tehtäviä. Tällaisia tehtäviä ovat esimerkiksi erilaisten matemaattisten teorioiden todistaminen tai shakin pelaaminen. (Copeland 2023.)

Ihmisten älykkyyttä harvoin arvostellaan yhden tietyn ominaisuuden avulla, vaan sitä pidetään monen keskenään erilaisen taidon yhdistelmänä. Erään teorian mukaan älykkyys voidaan jakaa kahdeksaan erilaiseen älykkyyteen: visuaalisavaruudellinen, keholliskinesteettinen, musiikillinen, kielellinen, loogismatemaattinen, intrapersonallinen, interpersoonallinen, naturalistinen. Teorian mukaan yksi yleinen älykkyys ei voisi millään selittää ihmisen kykyä toimia älykkäästi mitä erilaisemmissa tilanteissa. Näin käytössä olisi siis useisiin eri tehtäviin erikoistuneita ja toisistaan riippumattomia älykkyyksiä. Tekoälyn tutkimus on keskittynyt lähinnä muutamaosa-alueeseen edellä mainituista. Oppiminen, järkeily, ongelmanratkaisu, havaitseminen ja kielen käyttö ovat alueita, joissa tekoäly on vahvimmillaan. (Copeland 2023.)

### 2.2 Koneoppiminen

Koneoppiminen ja tekoäly ovat kaksi käsitettä, joita usein käytetään samassa asiayhteydessä tai vaihtokelpoisesti. Koneoppiminen on kuitenkin tekoälyn osa-alue, joka keskittyy kehittämään järjestelmiä, jotka voivat oppia automaattisesti ilman ihmisen väliintuloa. Koneoppimista käytetään pääasiassa suurten

datamäärien hallitsemiseen. Tavoitteena on löytää näistä datamassoista toistuvia kaavoja ja tehdä dataa koskevia päätöksiä. Oppiminen tulee mukaan siinä vaiheessa, kun järjestelmä oppii lisää kokemuksen ja lisädatan avulla ilman ihmisen avustusta. Tämän oppimisen kautta järjestelmä pystyy sitten tekemään parempia päätöksiä. Tietokoneiden ohjelmoijat ja sovellusten kehittäjät käyttävät koneoppimista muiden menetelmien ohella erilaisten tekoälyjärjestelmien kehityksessä, mistä johtuu termien läheinen suhde. (IBM 2023a.)

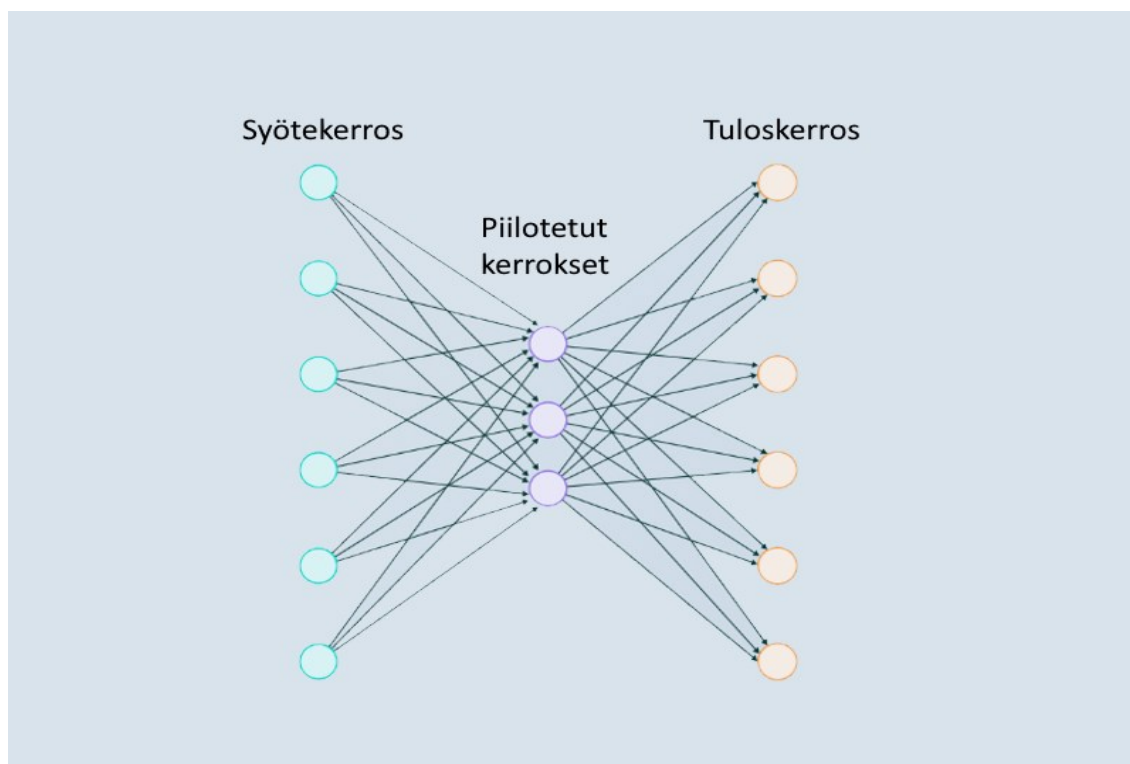
Koneoppiminen jaetaan kolmeen pääluokkaan: ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen. Ohjatussa oppimisessa järjestelmä pyrkii ennustamaan tai luokittelemaan tietoa esimerkkien perusteella. Ohjattu oppiminen käyttää koulutustietoa, joka opettaa malleja tuottamaan haluttua tulosta. Tämä tietojoukko sisältää syötteitä ja oikeita tuloksia, joka sallii mallin oppimisen ajan kanssa. Ohjaamaton oppiminen on edellä mainitun vastakohta siinä mielessä, että sille ei anneta valmiiksi vastauksia tai esimerkkejä, vaan järjestelmän tulee löytää tietoa ilman näitä. Järjestelmä etsii datasta rakenteita ja kaavoja, jotka voisivat auttaa sitä ymmärtämään tietoa paremmin. Vahvistusoppiminen tarkoittaa järjestelmän koulutusta ympäristössä, jossa se oppii lähinnä yrityksen ja erehdyksen kautta. Menetelmä on osittain sama kuin ohjatussa oppimisessa, mutta järjestelmää ei kouluteta datan avulla, vaan se oppii tehdessään. Tavoitteena vahvistusoppimisessa on löytää paras käytäntö erilaisten ongelmien ratkaisemiseen, tämä löydetään useamman peräkkäisen onnistuneen lopputuloksen päätteeksi. (IBM 2023a).

### 2.3 Neuroverkot

Neuroverkot ovat osa koneoppimista, niiden nimi ja rakenne ovat saaneet inspiraationsa ihmisen aivoista. Ne yrittävät myös jäljitellä sitä tapaa, jolla ihmisen biologiset neuronit viestivät toisilleen. Koneoppimisessa niitä käytetään pääasiassa monimutkaisten datamallien oppimiseen ja tunnistamiseen. Neuroverkko koostuu useista yksiköistä ja kerroksista, jotka ovat yhteydessä

keskenään. Nämä kerrokset sisältävät ainakin syötekerroksen, useamman piilotetun kerroksen ja tuloskerroksen. (IBM 2023c.)

Jokainen neuroni muodostaa yhteyden seuraavaan ja jokaisella neuronilla on oma painoarvo sekä kynnys. Jos jonkin yksittäisen neuronin tulos ylittää tuon määritetyn kynnyksiarvon, kyseinen neuroni aktivoituu ja lähettää dataa verkon seuraavalle kerrokselle. Jos kynnystä ei ylitetä, ei dataa lähetetä seuraavalle kerrokselle. Näin laskenta etenee verkon kerrokselta toiselle. Jokaista yksittäistä neuronia voidaan ajatella omana lineaarisena regressiomallina, joka koostuu syötteestä, painoarvoista, kynnyksestä ja tuloksesta. Painoarvot auttavat määrittelemään erilaisten muuttujien tärkeyttä, suuremmat painoarvot luonnollisesti vaikuttavat enemmän tulokseen. Kaikki syötteet kerrotaan niitä vastaavilla painoarvoilla ja lasketaan yhteen. Tämän jälkeen tulos ajetaan aktivointifunktion läpi, joka päättää lopullisen tuloksen. Jos saatu tulos ylittää asetetun kynnyksen, kyseinen neuroni aktivoituu ja lähettää tiedon eteenpäin seuraavalle kerrokselle. Näin saadusta tuloksesta tulee seuraavan kerroksen syöte. (IBM 2023c.) Kuvassa 1 havainnollistettu neuroverkon rakennetta.



Kuva 1. Neuroverkon rakenne (IBM 2023c, muokattu).

Tämä on tärkeää opinnäytetyön aiheen kannalta, sillä GPT-3 (Generative Pre-trained Transformer, suom. kielimalli) oli julkaisuhetkellään suurin koskaan kehitetty neuroverkko (Brown ym. 2023).

## 2.4 Syväoppiminen

Syväoppiminen on koneoppimisen alakategoria, jossa käytetään syviä neuroverkkoja monimutkaisten tietomallien oppimiseen ja ennustamiseen. Siitä on tullut laajimmin käytetty koneoppimisen lähestymistapa, koska se on saavuttanut erinomaisia tuloksia useissa monimutkaisissa kognitiivisissa tehtävissä, usein jopa ylittäen ihmisten suorituskyvyn. Yksi syväoppimisen suurimmista eduista on sen kyky oppia valtavia tietomääriä. Menestystä on saavutettu erityisesti kyberturvallisuudessa, luonnollisen kielen käsittelyssä, bioinformatiikassa, robotiikassa sekä lääketieteellisen tiedon käsittelyssä. (Alzubaidi, L. ym. 2021.)

Syväoppimisen ja koneoppimisen ero tulee siitä millaisen datan kanssa nämä työskentelevät ja mitä metodeja käytetään oppimiseen. Koneoppiminen käyttää jäsenneltyä ja merkittyä dataa ennusteiden tekemiseen. Syväoppiminen poistaa osan tietojen esikäsittelystä, joka osaltaan helpottaa prosessia huomattavasti. Nämä algoritmit voivat ottaa vastaan sekä käsitellä jäsentämätöntä dataa, kuten kuvia ja tekstiä. Se myös automatisoi ominaisuuksien poimimisen, jolloin riippuvuus ihmisten asiantuntijoiden avulle vähenee. Jos meillä olisi esimerkiksi joukko kuvia eläimistä ja haluaisimme luokitella ne lajin mukaan, syväoppimisen algoritmit voivat määrittää, mitkä ominaisuudet näistä eläimistä ovat tärkeimpiä erottamaan ne toisistaan. Koneoppimisessa tämä ominaisuuksien hierarkia täytyy määritellä manuaalisesti asiantuntijan toimesta. (IBM 2023b.)

GPT-3 käyttää syväoppimista muuntaja-arkkitehtuurinsa kouluttamiseen, joka taas mahdollistaa luonnollisen kielen käsittelyn ja kokonaisten viestien ymmärtämisen. Muuntajat on suunniteltu käsittelemään peräkkäistä

syöttödataa, kuten GPT-3 tapauksessa luonnollista kieltä. Muuntajat käsittelevät koko syötteen kerralla. (Brown ym. 2023.)

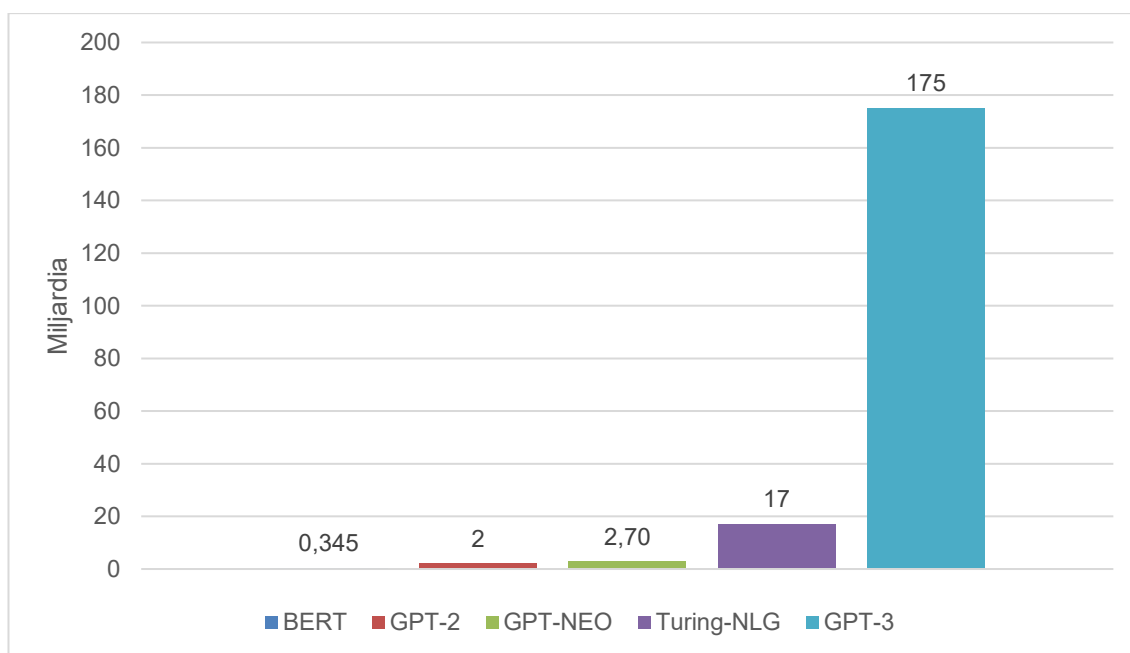
## 2.5 GPT ja kielimalli

GPT (Generative Pre-trained Transformer) on OpenAI:n kehittämä syväoppimiseen perustuva tekstinluonnin algoritmi. GPT perustuu aiemmin mainittuun transformer-arkkitehtuuriin, jonka avulla se pystyy tuottamaan hyvin luonnollisen näköistä tekstiä. Se on koulutettu ennustamaan mikä seuraava merkki voisi olla perustuen edellisiin merkkeihin. GPT-3 version kouluttamisessa käytettiin 175 miljardia parametria, mikä on kymmenen kertaa enemmän kuin millään aikaisemmalla kielimallilla. Valtavan opetusdatan avulla se on oppinut ymmärtämään kielioppia, semantiikkaa, kontekstia ja monia muita tekstin piirteitä. GPT:n suurin etu onkin juuri sen kyky sopeutua ja kehittyä jatkuvasti uuden opetusdatan avulla. Tämän ansiosta GPT:llä on huomattavasti potentiaalia monilla eri teknologian aloilla. (Brown ym. 2020.)

Vaikka GPT kuulostaa paperilla liian hyvältä ollakseen totta, ei sekään toimi ongelmattomasti. Aiemmin mainittu koulutusdatan raja-alue vuoteen 2021 GPT-3 versiossa on yksi suurimmista esteistä, tämä ongelma kuitenkin mitä todennäköisimmin tulee poistumaan tulevista versioista. Algoritmi voi myös usein tuottaa täysin väärää tietoa. Vaarallisuus piilee siinä, että se esittää tietonsa hyvinkin vakuuttavasti, mikä voi johtaa monta käyttäjää harhaan. Koska GPT rakentaa vastauksensa merkki kerrallaan, se käytännössä saa vastauksensa valmiiksi usean arvauksen jälkeen. Tämä osaltaan selittää, miksi vastaus voi usein olla väärä. Kuitenkin uudemmissa versioissa on hyvin mahdollista, että tätä ongelmaa saadaan huomattavasti korjattua. Koska GPT-3 on koulutettu internetistä löytyvällä tiedolla, on tiedon mukana tullut myös kaikki puolueellisuus ja ennakkoluulot. OpenAI tiedostaa tämän ongelman, ja mahdollisia syrjiviä vastauksia pyritään välttämään koulutusdatan tarkemman valikoinnin avulla. (Brown ym. 2020; Alzubaidi, L. ym. 2021.)

Kielimalli on tekstinluonnin algoritmi, joka on suunniteltu oppimaan kielen rakennetta ja kontekstia käyttäen suuria määriä opetusdataa. Kielimalli käyttää useimmiten neuroverkkoja tai muita syväoppimiseen perustuvia menetelmiä oppiakseen kielen käyttöä. Sen tärkein tavoite on yrittää ymmärtää, miten lauseen sanat liittyvät toisiinsa ja mikä on lauseiden välissä olevien sanojen merkitys. Aivan kuten ihmisen opetellessa äidinkieltään, kielimallin tulee ymmärtää kieliopin sääntöjä, sanastoa ja lauseiden rakennetta. Näitä taitoja yhdistämällä kielimalli onnistuu tuottamaan hyvin monipuolista ja sujuvaa tekstiä erilaisilla aihealueilla. (Brown ym. 2020.)

Tyypillisesti kielimalli jaetaan kahteen eri luokkaan: luokittelevaan ja generoivaan. Luokittelevaa mallia käytetään ennustamaan seuraava sana, kun generoiva malli kykenee tuottamaan uutta tekstiä. Generoiva malli voi myös tuottaa tekstiä täysin vapaasti tai annetun kontekstin perusteella, kuten GPT-3:a käytettäessä. Kielimallit ovat olleet mukana erilaisten chatbottien, puheentunnistuksen, konekääntämisen ja tiedonhaun kehittämisessä. GPT-3 on yksi nykypäivän kehittyneimmistä ja monipuolisimmista kielimalleista. (Brown ym. 2020.) Kuvassa 2 olevassa kaaviossa on vertailua parametrien määrässä eri kielimallien välillä.



Kuva 2. Kielimallien parametrien määrä miljardeissa.

### 3 GPT-3.5:n testaus

Tässä luvussa käydään läpi työkalun erilaisia mahdollisuuksia, joita ohjelmoija voi hyödyntää jokapäiväisissä tehtävissä. Helppoin tapa testata työkalun tehokkuutta olisi käyttää maksullista ohjelmointirajapintaa ja suorittaa komentoja suoraan koodieditorissa, mutta tämä opinnäytetyö on toteutettu täysin ilmaisilla vaihtoehtoilla. Visual Studio Coden lisäosista löytyy monta hyvää vaihtoehtoa, jos haluaa kokeilla, miltä tekoälyapuri tuntuu suoraan koodieditorissa ilman ylimääräisen verkkosivun käyttöä. Keväällä 2023 suosituimmat lisäosat ovat latausmääriä ja arvosanoja tarkastellessa "ChatGPT – Genie AI" sekä "Code GPT". Näistä lisäosista parhaat antavat samankaltaisen keskustelumahdollisuuden kuin verkkosivulla käytettävä versio. Niitä voi ajatella kuin ne olisivat tiimikavereita, joilta on helppoa kysyä neuvoa tai ohjeistusta projektin edetessä.

Työkalulle annetut syötteet, näillä saadut vastaukset ja Visual Studio Codessa tehdyt testaukset on havainnollistettu kuvissa. Työkalun kanssa voi kommunikoida myös suomeksi, mutta koska on ohjelmoinnista kyse, kanssakäyminen on tehty ohjelmoinnin standardikielellä eli englanniksi. Testauskehikko on koottu käymällä läpi ohjelmointiin liittyviä yleisiä tehtäviä ja kokoamalla yhteen näistä tärkeimpiä. Testeissä käydään läpi seuraavia asioita: koodin tuottaminen, vaihtoehtojen tuottaminen, esimerkkitiedon tuottaminen, datan muotoilu, koodin selittäminen, dokumentointi, virheiden tarkistus, virheiden korjaaminen, testien tuottaminen, koodin optimointi, koodin kääntö toiselle kielelle sekä tiedon etsiminen. Jos ChatGPT:ltä kysyy, millaisissa tehtävissä se voi olla ohjelmoijalle hyödyksi, tulee vastaukseksi samankaltainen, vaikka hieman lyhyempi lista.

Näitä ominaisuuksia hyödyntäessä on hyvä muistaa, että työkalun kanssa voi käydä jatkuvaa keskustelua, sillä se muistaa keskustelussa käytyt asiat. Koska tuotettu tieto ei aina pidä paikkaansa, on hyvä jatkaa keskustelua ja yrittää selvittää, voisiko asian tehdä toisin tai löytyykö muita tapoja korjata ongelma.

Työkalua voi myös suoraan huomauttaa, jos saatu tieto on selvästi väärää. (Brown ym. 2020.)

### 3.1 Koodin tuottaminen

Ensin testattiin puhdasta koodin tuottamista. Yhä harvemmin koodia täytyy tai kannattaa kirjoittaa tyhjästä, useimmiten sama asia on kuitenkin jo tehty toisaalla toiseen tarkoitukseen. Työkalua pyydettiin luomaan lista, jossa on satunnainen määrä satunnaisia hedelmiä (kuitenkin vähemmän kuin kymmenen hedelmää). Sen jälkeen tulostetaan lista, siinä olevien hedelmien määrä sekä lista käännettynä toisinpäin. Työkalulle annettu syöte näkyy kuvassa 3.

```
Can you create a program in python that has a list with random amount of random fruits in it. There should still be less than 10 fruits in total. It should first print out the list, then print how many items the list has, then print out the list reversed.
```

Kuva 3. Työkalulle annettu syöte.

Kuvassa 4 näkyy ChatGPT:n antama vastaus edellä mainittuun syötteeseen. Vastauksen lisäksi työkalu on lisännyt koodiin kommentteja koodin toiminnasta sekä koodin osien mukana tulee usein selityksiä annetun vastauksen toiminnasta, tämä selitys näkyy kuvassa 5.



```

import random

# Create a list of random fruits
fruits = random.sample(['apple', 'banana', 'cherry', 'date', 'elderberry',

# Print the list of fruits
print("List of fruits:", fruits)

# Print the number of items in the list
print("Number of items:", len(fruits))

# Print the reversed list of fruits
print("Reversed list of fruits:", fruits[::-1])

```

Kuva 4. Koodin tuottaminen.

In this program, we use the `random.sample` function to create a list of random fruits with a random length between 1 and 9. We then print the list of fruits, the number of items in the list (which we get using the `len` function), and the reversed list of fruits (which we get by slicing the list with `[::-1]`).

Kuva 5. Tuotetun koodin selitys.

Tässä vaiheessa koodin voi kopioida suoraan haluamaansa koodieditoriin, tässä opinnäytetyössä käytetään Visual Studio Codea. Kuvassa 6 näkyy tuloste, kun koodia on ajettu useamman kerran peräkkäin.

```

List of fruits: ['honeydew', 'date', 'cherry', 'grape', 'apple']
Number of items: 5
Reversed list of fruits: ['apple', 'grape', 'cherry', 'date', 'honeydew']
PS C:\Opiskelu\Opinnäytetyö\Koodit> & C:/Ohjelmat/Python/python.exe c:/Opiskelu/Opinnäy
List of fruits: ['elderberry', 'honeydew', 'apple', 'fig', 'cherry', 'grape']
Number of items: 6
Reversed list of fruits: ['grape', 'cherry', 'fig', 'apple', 'honeydew', 'elderberry']

```

Kuva 6. Tuotetun koodin ajaminen.

Tuotettu koodi toimii siis täysin odotetulla tavalla. ChatGPT loi ensin kymmenestä hedelmästä esimerkkidatan, josta sitten valittiin satunnainen määrä hedelmiä ja tehdään loput tulosteet. Pyydetty koodi oli melko yksinkertainen, mutta näyttää hyvin millaista tulosta saadaan aikaan, kunhan syöte on tarpeeksi tarkka. Syötteen tulee olla hyvin tarkka, jos haluaa saada nopeasti oikean ratkaisun, mutta täysin toisenlainen lähestymistapa on myös mahdollinen. Kuvassa 7 näkyy tulos, kun työkalua on pyydetty luomaan täysin satunnainen ohjelma. Myös satunnainen ohjelma toimii oikein.

```

import random

# Generate a random number between 1 and 10
secret_number = random.randint(1, 10)

# Initialize the number of guesses to 0
num_guesses = 0

# Loop until the user guesses the correct number
while True:
    # Ask the user to enter a guess
    guess = int(input("Guess the secret number (between 1 and 10): "))

    # Increment the number of guesses
    num_guesses += 1

    # Check if the guess is too high or too low
    if guess > secret_number:
        print("Too high! Try again.")
    elif guess < secret_number:
        print("Too low! Try again.")
    else:
        # The user guessed correctly
        print("Congratulations, you guessed the secret number in", num_guesses)
        break

```

This program uses the `random` module to generate a random number between 1 and 10, and then uses a `while` loop to repeatedly ask the user to guess the number until they guess correctly. Inside the loop, it checks if the guess is too high or too low, and provides feedback to the user. When the user guesses correctly, the loop is exited and the program prints a congratulatory message along with the number of guesses it took the user to guess the secret number.

```

Guess the secret number (between 1 and 10): 7
Too high! Try again.
Guess the secret number (between 1 and 10): 5
Too high! Try again.
Guess the secret number (between 1 and 10): 3
Too high! Try again.
Guess the secret number (between 1 and 10): 2
Congratulations, you guessed the secret number in 4 guesses!

```

Kuva 7. Satunnainen tuotettu koodi, selitys sekä tuloste.

Usein tuotettu vaihtoehto ei välttämättä toimi sellaisenaan tai sovellu kokonaisratkaisuun, johon sitä ollaan lisäämässä. Tällöin voidaan käyttää työkalun "Regenerate response"-komentoa, joka ottaa saman syötteen, mutta tuottaa erilaisen vastauksen. Parempi vaihtoehto on kuitenkin tehdä uusi syöte, jossa vain pyytää vaihtoehtoista ratkaisua. Koko ohjetta ei tarvitse syöttää uudelleen, sillä alkuperäinen syöte ja sen vastaus on työkalulla muistissa. Syötteeksi riittää yksinkertainen pyyntö "Could you provide an alternative solution?", kuvassa 8 näkyy vaihtoehtoinen koodi. Ohjelman idea pysyy kuitenkin täysin samana.

```
import random

# Define a list of fruits
fruit_list = ['apple', 'banana', 'cherry', 'date', 'elderberry', 'fig', 'grape']

# Generate a random number between 1 and 9 to determine the length of the list
list_length = random.randint(1, 9)

# Create a list of random fruits with the specified length
random_fruits = random.sample(fruit_list, list_length)

# Print the list of fruits
print("List of fruits:", random_fruits)

# Print the number of items in the list
print("Number of items:", len(random_fruits))

# Print the reversed list of fruits
print("Reversed list of fruits:", random_fruits[::-1])
```

Kuva 8. Vaihtoehtoinen ratkaisu.

Tällä kertaa hedelmien listasta valitaan arvotun numeron avulla tietty määrä hedelmiä. Tämän jälkeen luodaan lista, jossa on arvotun numeron verran satunnaisia hedelmiä. Lopuksi vielä samat toimenpiteet kuin aikaisemmin.

### 3.2 Esimerkkidatan tuottaminen

Esimerkkidatan luominen voi olla hyvinkin aikaa vievä prosessi, jos sitä ei ole valmiina tai helposti saatavilla. Vaikka tällaisen datan luontiin löytyy verkosta useita työkaluja, vaativat ne usein kirjautumista eivätkä välttämättä tarjoa kaikkia vaihtoehtoja, joita projekti tarvitsee. Kuvassa 9 näkyy tulos, kun työkalua on pyydetty luomaan esimerkkidataa kolmesta käyttäjästä. Käyttäjillä tulee olla etunimi, sukunimi, puhelinnumero sekä sähköpostiosoite.

```
contacts = [  
  {  
    'first_name': 'John',  
    'last_name': 'Doe',  
    'phone_number': '555-1234',  
    'email': 'john.doe@example.com'  
  },  
  {  
    'first_name': 'Jane',  
    'last_name': 'Smith',  
    'phone_number': '555-5678',  
    'email': 'jane.smith@example.com'  
  },  
  {  
    'first_name': 'Bob',  
    'last_name': 'Johnson',  
    'phone_number': '555-9012',  
    'email': 'bob.johnson@example.com'  
  }  
]
```

Kuva 9. Tuotettu esimerkkidata.

Jos syöte ei ole ollut tarpeeksi tarkka, voi olla, että vastaus tulee muodossa, jota ei voida suoraan käyttää. Koska keskustelu ja edelliset vastaukset pysyvät työkalulla muistissa, voidaan saatua esimerkkidataa muokata halutulla tavalla. Kuvassa 10 näkyy edellinen data, mutta nyt siihen on pyydetty lisäämään myös katuosoite.

```
contacts = [  
  {  
    'first_name': 'John',  
    'last_name': 'Doe',  
    'phone_number': '555-1234',  
    'email': 'john.doe@example.com',  
    'address': '123 Main St'  
  },  
  {  
    'first_name': 'Jane',  
    'last_name': 'Smith',  
    'phone_number': '555-5678',  
    'email': 'jane.smith@example.com',  
    'address': '456 Oak Ave'  
  },  
  {  
    'first_name': 'Bob',  
    'last_name': 'Johnson',  
    'phone_number': '555-9012',  
    'email': 'bob.johnson@example.com',  
    'address': '789 Pine Rd'  
  }  
]
```

Kuva 10. Muokattu esimerkkidata.

ChatGPT on myös kätevä työkalu jo olemassa olevan datan muokkaamiseen. Seuraavassa esimerkissä on kuvassa 10 olevien ihmisten tiedot, mutta tiedot ovat listassa peräkkäin. Jos datan määrä kasvaisi tästä huomattavasti suuremmaksi, olisi sen järjestely manuaalisesti hyvin aikaa vievää ja työlästä. Kuvassa 11 näkyy annettu syöte sekä vastauksena saatu jäsennelty data.

I have a list of people and their information, could you help organize this list ready for programming?

John Doe, 555-1234, john.doe@example.com, 123 Main St, Jane Smith, 555-5678, jane.smith@example.com, 456 Oak Ave, Bob Johnson, 555-9012, bob.johnson@example.com, 789 Pine Rd

```
people = [  
  {  
    "name": "John Doe",  
    "phone": "555-1234",  
    "email": "john.doe@example.com",  
    "address": "123 Main St"  
  },  
  {  
    "name": "Jane Smith",  
    "phone": "555-5678",  
    "email": "jane.smith@example.com",  
    "address": "456 Oak Ave"  
  },  
  {  
    "name": "Bob Johnson",  
    "phone": "555-9012",  
    "email": "bob.johnson@example.com",  
    "address": "789 Pine Rd"  
  }  
]
```

Kuva 11. Annettu syöte ja aremmin järjestetty esimerkkidata.

### 3.3 Koodin selittäminen

Vaikka ChatGPT voi tuottaa koodia mitä erilaisempiin tarpeisiin, on tärkeää ymmärtää, mitä tuotettu koodi oikeasti tekee. On hyvin vaarallista tietoturvan ja sovelluksen toimivuuden kannalta, jos tekemäänsä koodia ei ymmärrä. Ohjelmistotalalla usein vitsaillaan siitä, että tehtyä koodia ei täysin ymmärretä tai että ei ole tiedossa miksi koodi toimii tai ei toimi. Vanhempaa koodia katsottaessa sen joukosta voi usein löytyä kommentteja, jotka kertovat, ettei

kehittäjällä ollut mitään tietoa mitä seuraavassa osassa tapahtuu. Tämä on aivan todellinen ongelma, eikä siitä ole mahdollista päästä koskaan täysin eroon. Vaikka kokemusta löytyisi kymmeniä vuosia, ala liikkuu niin hurjaa vauhtia eteenpäin ettei uusimmista teknologioista ole mahdollista tietää kaikkea. JavaScript-ohjelmointikielen ekosysteemi kehittyy tällä hetkellä niin nopeasti, että moni on väsynyt yrittämään pysyä kärryillä uusimmista käänteistä.

Yksi ChatGPT ominaisuuksista on selittää sille syötteenä annettua koodia. Tämä on tavallaan sama tilanne kuin pyytäisi luennolla opettajaa kertomaan lisää siitä, miten kirjoitettu koodi toimii. Apu toimii myös niin päin, että pyytää työkalua kertomaan tuottamansa koodin toiminnasta. Otetaan esimerkiksi aiemmin laadittu satunnainen koodi, jossa työkalu loi pelin, jossa käyttäjä arvailee numeroa, kunnes arvaus osuu oikeaan. Kuvassa 12 työkalua on pyydetty selittämään luomansa koodi mahdollisimman yksityiskohtaisesti. Vastauksena saadaan yksityiskohtainen ja askel kerrallaan tehty selitys koodin toiminnasta.

1. First, we import the `random` module, which provides functions for generating random numbers.

python

Copy code

```
import random
```

2. Next, we use the `random.randint()` function to generate a random integer between 1 and 10, inclusive. This will be the "secret number" that the user has to guess.

python


Copy code

```
secret_number = random.randint(1, 10)
```



3. We initialize a variable `num_guesses` to keep track of how many guesses the user has made so far.


python

 Copy code

```
num_guesses = 0
```

4. We use a `while` loop to repeatedly ask the user to guess the secret number, until they guess correctly. The loop runs indefinitely, until we use the `break` statement to exit the loop when the user guesses correctly.


python

 Copy code

```
while True:
    # code here
    if condition:
        # code here
    elif condition:
        # code here
    else:
        # code here
    break
```

5. Inside the loop, we use the `input()` function to ask the user to enter a guess. We then convert their input to an integer using the `int()` function.


python

 Copy code

```
guess = int(input("Guess the secret number (between 1 and 10): "))
```

6. We increment the `num_guesses` variable by 1 each time the user makes a guess.

python

 Copy code

```
num_guesses += 1
```

7. We check if the user's guess is too high or too low compared to the `secret_number`, and provide feedback to the user. If the guess is correct, we break out of the loop using the `break` statement.

python

Copy code

```
if guess > secret_number:
    print("Too high! Try again.")
elif guess < secret_number:
    print("Too low! Try again.")
else:
    print("Congratulations, you guessed the secret number in", num_guesses,
    break
```

8. When the loop is exited (because the user guessed the secret number correctly), we print a congratulatory message along with the number of guesses it took the user to guess the secret number.

python

Copy code

```
print("Congratulations, you guessed the secret number in", num_guesses, "gues
```

Kuva 12. Selitys koodin toiminnasta.

### 3.4 Koodin dokumentointi

Koodin dokumentointi on yksi erittäin tärkeä osa ohjelmointia. Se toimii eräänlaisena käyttöohjeena kaikille niille, jotka tulevat työskentelemään projektin parissa alkuperäisen kehityksen jälkeen. Koska projektit usein ovat useamman kuin yhden henkilön tuotoksia, on hyvin tärkeää, että dokumentointi on kunnossa, jotta koko tiimin olisi helpompaa ja tehokkaampaa työskennellä projektin parissa. Projektit myös harvemmin ovat pysyvästi yhden tai muutaman henkilön työ, ihmiset siirtyvät toisiin työpaikkoihin tai uusiin projekteihin ja vanha projekti jää jonkun toisen hoidettavaksi. Käyttöohjeen tarve on helppo ymmärtää, kun ajattelee minkä tahansa uuden ja tuntemattoman laitteen käyttöönottoa.

Työkalu voidaan pyytää dokumentoimaan koodia vähän samalla tavalla kuin siltä voi pyytää koodin selitystä. Nämä kaksi asiaa kuitenkin kulkevat melko lailla käsi kädessä. Syötteenä voidaan antaa joko täysin uutta koodia, tai pyytää työkalua dokumentoimaan aiemmin vastauksena saatua koodia. Kuvassa 13 työkalua on pyydetty dokumentoimaan mahdollisimman yksityiskohtaisesti aiemmin luodun numeroiden arvauspelin koodia.

```
# Import the "random" module to generate random numbers
import random

# Generate a random integer between 1 and 10, inclusive, and assign it to "secret_number"
secret_number = random.randint(1, 10)

# Initialize a variable to keep track of the number of guesses made by the user
num_guesses = 0

# Loop indefinitely until the user guesses the correct number
while True:
    # Ask the user to guess the secret number and convert their input to an integer
    guess = int(input("Guess the secret number (between 1 and 10): "))

    # Increment the number of guesses made by the user
    num_guesses += 1

    # If the user's guess is higher than the secret number, provide feedback
    if guess > secret_number:
        print("Too high! Try again.")

    # If the user's guess is lower than the secret number, provide feedback
    elif guess < secret_number:
        print("Too low! Try again.")

    # If the user's guess is equal to the secret number, provide feedback and break
    else:
        print("Congratulations, you guessed the secret number in", num_guesses, "guesses.")
        break
```

Kuva 13. Koodin dokumentointi.

### 3.5 Virheiden tarkistus

Virheiden tarkistaminen ja korjaaminen on yksi suurimmista aikaa vievistä osa-alueista ohjelmoinnissa. Ongelmia voi yrittää välttää noudattamalla erilaisia hyviä ohjelmointikäytäntöjä, mutta virheiden ilmaantuminen projektin jossain vaiheessa on lähes väistämätöntä. Hyvä suunnitteluvaihe, koodin dokumentointi, versiohallinta, modularisointi, testaaminen sekä erilaiset koodieditorien lisäosat ovat kaikki toimintatapoja, joilla pyritään tuottamaan virheetöntä koodia tai ainakin korjaamaan virheellistä koodia. Ongelma on siis selvästi iso sekä kohtuullisen väistämätön, jos sitä varten on kehitetty mitä erilaisempia työkaluja.

Monet erilaisista virheistä huomataan nykyään automaattisesti juuri edellä mainittujen työkalujen ansiosta. Pienistä kirjoitusvirheistä tai puuttuvista välimerkeistä tulee melkein heti huomautus, ja ne on helppo korjata kirjoitusvaiheessa. Moni automaattinen työkalu huomaa myös isompia ongelmia, kuten muuttujien vääräntyyppisiä tai puuttuvia arvoja. Kuitenkin ohjelman isommassa kokonaisuudessa ilmenevät ongelmat menevät läpi näistä tarkistuksista. Tämä johtuu pääosin siitä, etteivät automaattiset tarkistukset välttämättä vielä tiedä, mitä ohjelmalla on oikeasti tarkoitus tehdä. Jos koodi näyttää sääntöjen mukaan täysin oikealta, ei siitä myöskään tule automaattisesti virheilmoituksia.

Tällaiset ongelmat ilmenevät vasta siinä vaiheessa, kun koodia ryhdytään ajamaan. Onneksi myös lopuksi saatavien virheilmoitusten kuvaukset ovat usein todella hyviä ja osaavat kertoa täsmälleen, mitä täytyy muuttaa tai korjata. Joskus kuitenkin tulee vastaan tilanteita, jolloin avuksi täytyy ottaa virhekoodien ja tiedon etsiminen, tällaisessa tilanteessa ChatGPT voi olla hyödyksi. Sen käyttö onnistuu hyvin myös aiemmin mainituissa ongelmatilanteissa, mutta parhaiten se sopii juuri siihen tilanteeseen, kun ohjelmoijalla ei ole tarkkaa tietoa siitä, missä ongelma on. Muokataan aiemmin luotua numeroiden arvauspeliä siten, että koodi on virheellistä useammassa kohdassa. Poistetaan muutama merkki, muutetaan muuttujien nimiä ja katsotaan, mitä ChatGPT

antaa vastaukseksi, kun siltä kysytään, mikä koodissa on vikana. Kuvassa 14 näkyy saatu vastaus, kun syötteenä on pyydetty kertomaan, mikä koodissa on vialla, sekä mahdollisia korjausehdotuksia. Syötteeseen voi vapaasti lisätä koodin pätkiä, työkalu osaa tulkita koodia, vaikka sen muotoilu ei olisi enää täysin oikein. Hyödyllistä on myös kertoa työkalulle, mistä ohjelmointikielestä on kyse.

There are a few issues with this code:

1. The variable `num_guesses` is always initialized to -1, which means that the first time it is incremented, its value will be 0. It should be initialized to 0 instead.
2. The variable `guess` is being converted to a string using `str()`, which is unnecessary because the `input()` function already returns a string.
3. The `if` statement that checks if the guess is too high or too low is incorrect. It should check if `guess` is greater than or less than `secret_number`, respectively.

```
import random

# Generate a random number between 1 and 10
secret_number = random.randint(1, 10)

# Initialize the number of guesses to 0
num_guesses = 0

# Loop until the user guesses the correct number
while True:
    # Ask the user to enter a guess
    guess = input("Guess the secret number (between 1 and 10): ")

    # Increment the number of guesses
    num_guesses += 1

    # Check if the guess is too high or too low
    if int(guess) > secret_number:
        print("Too high! Try again.")
    elif int(guess) < secret_number:
        print("Too low! Try again.")
    else:
        # The user guessed correctly
        print("Congratulations, you guessed the secret number in", num_guesses)
        break
```

Kuva 14. Virheellisen koodin korjaus.

Vastauksena saadaan lähes sama versio kuin aiemmin luotu toimiva koodi. Ongelmat on selitetty yksi kerrallaan ja työkalu tarjoaa myös korjatun koodin ainakin silloin kun sitä on erikseen pyydetty. Yksi ilmaisen version rajoituksista voi tulla tässä kohtaa vastaan, sillä GPT-3.5 voi ottaa syötteeksi enintään noin 4000 merkkiä. Sama pätee sen antamaan vastaukseen. Sille siis ei voi syöttää kokonaista pitkää lähdekoodia kerralla, tässä kohtaa ohjelmoijan on hyvä osata erotella itse mahdolliset ongelmalliset osiot ja lähteä ratkomaan ongelmia sitä kautta. Koodia voi syöttää useamman syötteen avulla, mutta tästä voi aiheutua ylimääräisiä ongelmia, jos koodia joutuu jakamaan useampaan osaan. Tämäkin rajoitus on tosin jo poistunut uudemmissa versioista kuten GPT-4, joka osaa ottaa vastaan jo 32 000 merkkiä. Tekoälyn kehittyessä sellaisella vauhdilla kuin tällä hetkellä, edellä mainittu rajoitus tulee mitä luultavimmin myös poistumaan melko nopeasti (OpenAI 2023.)

### 3.6 Yksikkötestit

Yksikkötestit ovat sovelluksien testaamista, jossa testataan yksittäisiä sovelluksen komponentteja. Tämä testaus suoritetaan lähes aina kehitysvaiheessa ja sillä pyritään varmistamaan, että sovelluksen osat toimivat odotetulla tavalla. Yksikkötestit eristävät tiettyjä osia koodista ja varmistavat, että ne toimivat oikein. Tällainen yksittäinen osa voi olla esimerkiksi yksittäinen funktio, metodi, käytäntö, moduuli tai objekti. Peruskäytäntönä on luoda yksikkötestit omiin erillisiin tiedostoihinsa tai omaan testikehykseen.

Tämäntyyppisen testauksen tavoitteena on auttaa havaitsemaan koodin mahdollisia virheitä mahdollisimman aikaisessa vaiheessa. Tämä auttaa vähentämään ongelmista aiheutuneita kustannuksia ja helpottaa koodin korjaamista huomattavasti. Myöhäisemmässä vaiheessa huomattavat ongelmat ovat lähes aina vaikeammin korjattavia ja vievät huomattavasti enemmän aikaa. Testaamista varten on kehitetty myös kehyksiä ja kirjastoja, jotka automatisoivat suuren osan testaamisesta vähentäen manuaalista työtä. ChatGPT:tä voidaan

käyttää kirjoittamaan aiemmin mainittuja yksikkötestejä, otetaan tähän esimerkiksi alussa luotu koodi hedelmien listasta.

Kuvassa 15 näkyy vastauksena saatu testien koodi, testien selitys sekä koodin ajamisen tulos. Syötteenä työkalulle annettiin pyyntö yksikkötestien luomisesta ja ohjelman alkuperäinen koodi.

```
import unittest
import random

class TestFruits(unittest.TestCase):
    def setUp(self):
        self.fruits = random.sample(['apple', 'banana', 'cherry', 'date', 'elderberry'], 5)

    def test_fruits_list(self):
        self.assertIsInstance(self.fruits, list)
        for fruit in self.fruits:
            self.assertIn(fruit, ['apple', 'banana', 'cherry', 'date', 'elderberry'])

    def test_fruits_length(self):
        self.assertEqual(len(self.fruits), len(self.fruits[::-1]))

if __name__ == '__main__':
    unittest.main()
```

Explanation:

- `setUp`: This method is called before each test method, and it creates a new random list of fruits each time.
- `test_fruits_list`: This method checks that the `fruits` variable is a list and that each fruit in the list is one of the allowed fruits.
- `test_fruits_length`: This method checks that the length of the `fruits` list is equal to the length of the reversed `fruits` list.

```
..
-----
Ran 2 tests in 0.000s
OK
```

Kuva 15. Yksikkötestien koodi ja selitys.

Vastauksena saatu koodi tallennettiin erilliseen tiedostoon. ChatGPT loi metodin, jota kutsutaan ennen jokaista testimetodia. Tässä metodissa luotiin aina uusi satunnainen hedelmien lista. Ensimmäinen testimetodi tarkistaa, että hedelmälistan muuttuja on tyypiltään lista ja että jokainen listalla oleva hedelmä on yksi sallituista hedelmistä. Toinen testimetodi tarkistaa, että hedelmälistan pituus on sama kuin käänteisen listan pituus. Jos nämä kaikki testit menevät läpi, ohjelma toimii oletetulla tavalla ja testiohjelmaa ajettaessa tulostuu testien lukumäärä, testeihin käytetty aika sekä tulokset. Testien luominen näin yksinkertaiselle ohjelmalle on vielä helppoa, mutta laajemman ohjelman kanssa ChatGPT tarvitsee hyvin yksityiskohtaista tietoa koodin toiminnasta, ennen kuin se osaa tuottaa oikeat yksikkötestit.

### 3.7 Koodin optimointi

Optimoinnin suurimmat hyödyt ovat koodin suorituskyky ja luettavuus. Koodin hyvä suorituskyky siirtyy suoraan itse sovelluksen tai verkkosivun suorituskykyyn, tehokas toiminta taustalla varmistaa myös tehokkaan toiminnan etusivulla. Vaikka käyttäjä saa tästä suurimman hyödyn, on optimaalisesta koodista hyötyä myös itse kehittäjälle. Kaikki projektiin liittyvä testaaminen on huomattavasti helpompaa ja vähemmän aikaa vievää, kun projekti toimii tehokkaasti ilman vältettävissä olevia odotusaikoja. Nykypäivän odotus käyttäjän puolella on, että sovellus tai verkkosivu aukeaisi muutamissa sekunneissa. Jos käynnistys kestää tätä kauemmin, on hyvin todennäköistä, että käyttäjä jättää sovelluksen avaamatta tai siirtyy toiselle verkkosivulle.

Koodin luettavuuden parantaminen on myös tärkeä osa kehitysvaihetta, tähän liittyy aiemmissa luvuissa mainitut dokumentointi ja kommentointi. Jos saman asian saa tehtyä viidellä koodirivillä, miksi siitä tekisi silloin 30 riviä koodia. Tällaista optimointia voi olla vaikea toteuttaa itse, sillä näitä mahdollisuuksia on hankala huomata omassa koodissa ilman että sitä on tarkastellut pidemmän aikaa. Siksi on hyvä antaa esimerkiksi tiimin toisen jäsenen katsoa koodia läpi, uusi silmäpari saattaa huomata montakin kohtaa, jotka voisi toteuttaa tehokkaammin.



Toinen mahdollisuus on käyttää työkalua, kuten ChatGPT. Sille voi antaa syötteenä sovelluksen koodin ja kysyä, olisiko sitä mahdollista optimoida. Tässä opinnäytetyössä luodut esimerkkiohjelmat ovat melko yksinkertaisia, joten niistä ei löydy hirveästi paranneltavaa. Selvitetään kuitenkin, voisiko numeroiden arvauspeliä optimoida. Kuvassa 16 näkyy vastaus, kun syötteenä annettiin numeropelin alkuperäinen koodi ja pyydettiin optimointia.

1. Use a f-string instead of concatenation to print the result message:  

```
python Copy code  
  
print(f"Congratulations, you guessed the secret number in {num_guesses} guesses")
```
2. Validate the user input to avoid errors if they enter a non-integer value:  

```
python Copy code  
  
while True:  
    try:  
        guess = int(input("Guess the secret number (between 1 and 10): "))  
        break  
    except ValueError:  
        print("Invalid input. Please enter an integer.")
```
3. You could limit the number of guesses by adding a maximum number of attempts:  

```
bash Copy code  
  
MAX_GUESSES = 5  
  
while num_guesses < MAX_GUESSES:  
    ...  
print("Sorry, you did not guess the secret number in time.")
```

4. To improve the user experience, you could also add a prompt to play again:

```
python Copy code  
  
while True:  
    # Play the game  
    ...  
    play_again = input("Do you want to play again? (y/n): ")  
    if play_again.lower() != 'y':  
        break
```

Kuva 16. Ohjeet numeropelin optimointiin.

Työkalu antoi hyvin yksityiskohtaiset ohjeet mitä kaikkea numeropelistä voisi vielä muuttaa koodiesimerkkeineen. Ensimmäinen ehdotus koskee string-muuttujien yhdistämistä, siihen työkalu ehdottaa käytettään python-kielen f-string-metodia. Tämän tehokkuudesta voi olla montaa mieltä, mutta luettavuudeltaan se on parempi vaihtoehto. Toinen korjaus on tarkistaa käyttäjän syöttämä data, jos tämä antaa arvauksena jotain muuta kuin kokonaislukutyypin muuttujan. Käyttäjän syöttämä data tulisi tarkistaa aina kun se on mahdollista, tämä auttaa hyvin monen tietoturvaan liittyvän ongelman kanssa. Kolmantena ehdotuksena on rajoittaa arvausten määrä viiteen, jotta käyttäjä ei käy kaikkia numeroita läpi. Viimeiseksi työkalu ehdottaa käyttäjäkokemuksen parantamista sillä, että loppuun lisättäisiin kysymys haluaako käyttäjä pelata uudelleen.

Tuloksena oli siis hyvin käyttökelpoisia ehdotuksia ja korjauksia. Tämän esimerkin kohdalla näistä kaikki olisivat hyödyllisiä parannuksia. Isompien projektien parissa tällainen optimointi voi osoittautua hyvin hyödylliseksi. Usein työkalu näyttää korjausehdotukset, mutta ei lopullista koodia. Tässä tapauksessa, koska koodi on tarpeeksi lyhyt, työkalu antoi myös lopullisen koodin vastauksen lopussa, tämän koodin näkee kuvasta 17. Koodia optimoidessa täytyy kuitenkin pitää mielessä, että vähempi ei aina ole parempi. Jos koodin saa tiivistettyä muutamaan riviin, voi se tehdä koodin lukemisesta

paljon vaikeampaa, jos aiempi versio oli pidempi mutta helposti ymmärrettävissä.

```
import random

MAX_GUESSES = 5

while True:
    # Generate a random number between 1 and 10
    secret_number = random.randint(1, 10)

    # Initialize the number of guesses to 0
    num_guesses = 0

    # Loop until the user guesses the correct number or runs out of guesses
    while num_guesses < MAX_GUESSES:
        # Ask the user to enter a guess
        while True:
            try:
                guess = int(input("Guess the secret number (between 1 and 10): "))
                break
            except ValueError:
                print("Invalid input. Please enter an integer.")

        # Increment the number of guesses
        num_guesses += 1

        # Check if the guess is too high or too low
        if guess > secret_number:
            print("Too high! Try again.")
        elif guess < secret_number:
            print("Too low! Try again.")
        else:
```

Kuva 17. Osa optimoitua koodia.

### 3.8 Koodin kääntö toiselle kielelle

Joskus voi olla tarvetta kääntää koodia ohjelmointikieleltä toiselle. Vähänkin isomman projektin kanssa tämä ei ole yksinkertainen tehtävä ja se tulisi tehdä huolellisesti suunnitellen vaihe kerrallaan. Prosessissa tulisi olla paljon testausta, hyvin ymmärrystä molemmista kielistä sekä hyvä käsitys koodin arkkitehtuurista ja suunnittelusta. Ei siis kuulosta mitenkään kevyin perustein aloitettavalta projektilta. Pienemmissä määrin tämä voi kuitenkin olla melko vaivaton prosessi. Usein ongelmien ratkaisuja etsiessä saattaa törmätä tilanteeseen, jossa ongelma on ratkaistu mutta ratkaisu on tehty eri ohjelmointikielellä. Tätä alkaa tapahtua varsinkin silloin, kun ratkaisua ei löydy kovin helpolla vaan sitä joutuu etsimään huolellisesti.

ChatGPT voi olla avuksi näissä käännoksissä, jossa koodi pysyy työkalun rajoitteiden sisällä ja se tuntee käytetyn ohjelmointikielen. Jos työkalulta kysyy suoraan mitkä ohjelmointikielet ovat sille tuttuja, vastauksena saadaan seuraavanlainen lista: python, java, c++, javascript, php, ruby, swift, kotlin, typescript, go, rust, matlab, sql, html/css. Lista on siis melko kattava ja apua löytyy melkein kaikkiin käytetyimpiin ohjelmointikieliin. Käytetään tässä esimerkkinä aiemmin luotua hedelmälistan optimoimatonta koodia. Kuvassa 18 näkyy tulos, kun syötteenä annettiin hedelmälistan koodi ja pyydettiin kääntämään se PHP:lle. Koodi myös ajettiin tämän jälkeen komentokehoteessa ja se toimi oikein.

```

<?php
// Generate a random number between 1 and 10
$secret_number = rand(1, 10);

// Initialize the number of guesses to 0
$num_guesses = 0;

// Loop until the user guesses the correct number
while (true) {
    // Ask the user to enter a guess
    echo "Guess the secret number (between 1 and 10): ";
    $guess = trim(fgets(STDIN));

    // Increment the number of guesses
    $num_guesses++;

    // Check if the guess is too high or too low
    if ($guess > $secret_number) {
        echo "Too high! Try again.\n";
    } elseif ($guess < $secret_number) {
        echo "Too low! Try again.\n";
    } else {
        // The user guessed correctly
        echo "Congratulations, you guessed the secret number in $num_guesses guesses!\n";
        break;
    }
}
}

```

```

Guess the secret number (between 1 and 10): 5
Too high! Try again.
Guess the secret number (between 1 and 10): 2
Congratulations, you guessed the secret number in 2 guesses!

```

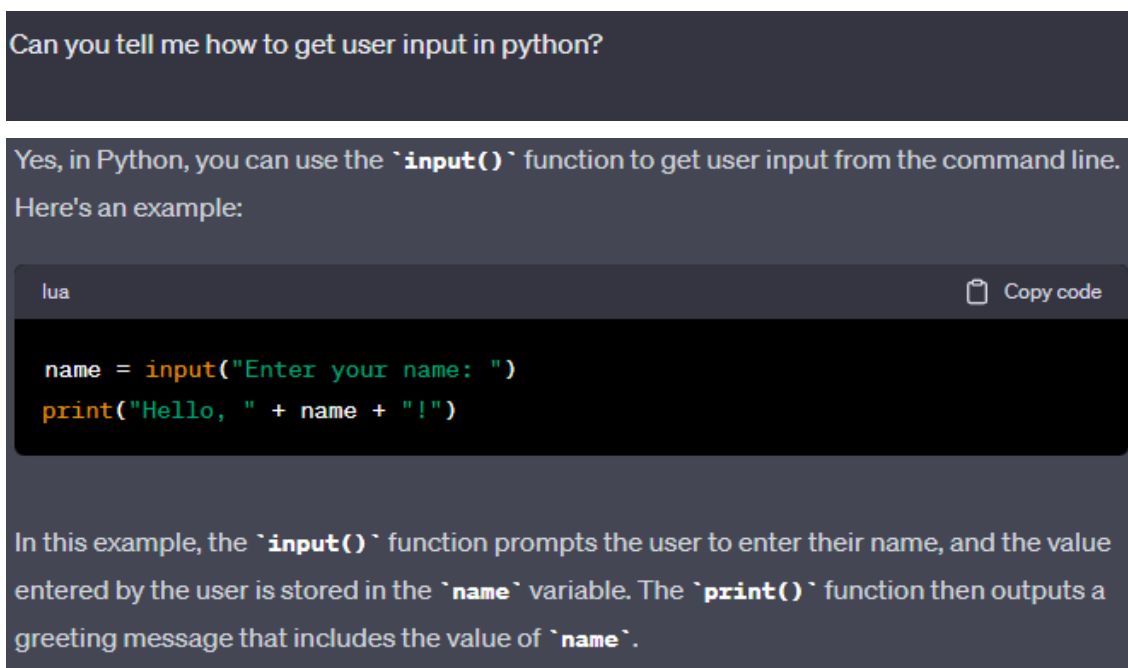
Kuva 18. Toiselle kielelle käännetty koodi.

### 3.9 Tiedon etsiminen

Yksi suuri osa ohjelmointia on jatkuva ratkaisujen ja tiedon etsiminen erilaisista verkkolähteistä ja dokumentaatioista. Ohjelmointikielen kokonaisen syntaksin muistaminen on ihmiselle lähes mahdotonta. Vaikka kielen tuntisi läpikohtaisin vuosien työkokemuksen jälkeen, kehitys tuo jatkuvasti mukanaan uusia asioita, jotka täytyy opetella erikseen. Tähän liittyy vahvasti aiemmin mainittu väsyminen jatkuvaan kehitykseen ja muutokseen. Hakukoneiden kehityksen myötä oikean tiedon löytäminen on tullut jo melko helpoksi, mutta tekoälyn myötä tämä prosessi tulee muuttumaan vielä vahvasti. Ajan säästäminen on huomattavaa, kun vastausta ei tarvitse enää etsiä, vaan täytyy vain tietää

tarkalleen, mitä kysyy. Tässä kohtaa on hyvä muistaa, että GPT-3.5 kuitenkin rajoittuu vuoteen 2021, joten uusimpien teknologioiden tai tapahtumien kanssa siitä ei ole apua. On myös hyvä pitää mielessä, että se saattaa antaa täysin väärää tietoa. Uusimmat versiot näistä tekoälypureista osaavat selata verkkosivuja, joten niillä on aina täysi pääsy uusimpaan tietoon. Ne myös osaavat antaa käyttäjälle tiedon lähteen, jotta käyttäjän on helpompi itse päättää voiko tietoon luottaa vai ei. (OpenAI 2023.)

Sen sijaan että hakusanat kirjoitettaisiin hakukoneeseen, voi ne nyt kirjoittaa suoraan työkalulle syötteenä. Suurimpana etuna tässä on, että kun vastaus on löytynyt, voi sitä helposti kehittää eteenpäin antamalla työkalulle osia omasta koodista ja näin selvittää, kuinka vastausta voi käyttää juuri oman projektin kohdalla. Kuvassa 19 työkalua on pyydetty selittämään, kuinka pythonia käytettäessä voidaan ottaa talteen käyttäjän syöttämä tieto. Kysymys on yksinkertainen, mutta vain mielikuvitus on rajana kysymyksiä esitettäessä. Käytössä on kuitenkin valtava määrä internetiin tallennettua tietoa.



Kuva 19. Tiedon etsiminen.

## 4 Tulevaisuuden näkymiä ja johtopäätöksiä

Tekoälyn tulevaisuutta ja sen vaikutuksia on hyvin vaikeaa ennustaa tarkasti. Erilaisia ennusteita voidaan laatia, mutta viime vuosina tapahtunut kehitys on näyttänyt selvästi, ettei niitä pystytä tällä hetkellä laatimaan tarkasti. Vaikka ennusteet osuisivat oikeaan, niiden aikataulua on hyvin hankala määritellä. Jo nyt monet kehityksen isommat askeleet ovat tapahtuneet paljon nopeammin kuin mitä odotettiin. Tekoäly koskettaa jo hyvin monta osa-aluetta ihmisten jokapäiväisessä elämässä, vaikka suuri osa näistä työkaluista ei olekaan millään tavalla näkyviä tavalliselle käyttäjälle (OpenAI 2023). Tässä luvussa keskitytään kuitenkin katsomaan niitä mahdollisuuksia, joita tekoäly voisi tuoda mukanaan ohjelmointiin. Käydään myös läpi muutamia jo olemassa olevia työkaluja, jotka ovat lähes suoraa seurausta viime vuosien huimasta kehityksestä.

Luvussa 3 käytiin läpi erilaisia ohjelmoinnin osa-alueita, joissa tekoäly osaa jo nyt avustaa huomattavasti. Testeistä huomattiin, että vaikka käytössä oli yksi ensimmäisistä vapaasti saatavilla olevista työkaluista, se osaa auttaa jo lähes kaikissa ohjelmoinnin osa-alueissa. Tämä varmistaa jo sen, että ohjelmoinnin työnkuva tulee muuttumaan huomattavasti ja näiden työkalujen osaaminen tulee olemaan aivan yhtä tärkeää kuin tiedon etsiminen on ollut tähän asti. Jo nyt saatavilla oleva GPT-4-versio osaa lähes kaikkia testeissä läpi käytyjä asioita paremmin kuin testattu GPT-3.5 (OpenAI 2023). Ohjelmoijan työ siis tulee ainakin helpottumaan ja muuttumaan hyvinkin paljon, kuitenkin taustalla tulee säilymään tarve osata ohjelmointia myös itse. Tekoälyn tehostamat työkalut tuskin tulevat täysin korvaamaan ohjelmoijia, koska niiltä täytyy osata pyytää asioita tietyllä tavalla. Syötteen antaminen on hyvin yksinkertaista, mutta tarvittu tuloksen saamiseksi tulee käyttäjällä olla kokemusta ohjelmoinnista. On myös tärkeää, että saatu tulos osataan tarkistaa asiantuntevasti, jotta käyttöön ei tule esimerkiksi tietoturvallisesti vaarallista koodia.

Yksi kysymys kuuluu, pystyykö tekoäly jossain vaiheessa kirjoittamaan koodia täysin itsenäisesti. Tämä ajatus on varmasti käynyt lähes jokaisen sellaisen ohjelmoijan mielessä, joka seuraa tekoälyn kehitystä aktiivisesti. Jos tällainen olisi mahdollista, onko myös mahdollista, että ohjelmoijat voidaan täysin korvata? Monen mielestä varmasti huono uutinen onkin, että tämä kehitysaskel on jo ohitettu. Vuoden 2023 alkupuolella ilmestyi Auto-GPT, joka pyrkii tekemään aiemmin mainitusta GPT-4:stä täysin itsenäisen, ja projekti onkin jo osittain onnistunut siinä. Työkalu on kielimalli, jolle voidaan antaa tehtäviä ja niiden suorittamiseen tiettyjä työkaluja. Työkalun täytyy sitten itsenäisesti selvittää, kuinka tehtävä voidaan ratkaista annetuilla työkaluilla. (AutoGPT 2023.)

Jos tekoäly kykenee lähivuosina tuottamaan käytettävää koodia täysin itsenäisesti, tuo se mukanaan myös useita haasteita. Kehityksen liikkuesssa hurjaa vauhtia ja lainsäädännön tullessa useita vuosia perässä miten pystytään varmistamaan, että luotu koodi on linjassa ihmisten etiikan ja moraalisten arvojen kanssa tai että koodi ei ole suoraan vahingollista ihmisille? Näitä vapaasti saatavilla olevia työkaluja voidaan helposti käyttää myös haitallisiin tarpeisiin, kuten haittaohjelmia kehitettäessä tai huijausviestejä laatiessa. Mahdollisuutena täytyy myös pitää sitä, että käytössä oleva työkalu olisi saastutettu haittaohjelmilla ja se olisi laitettu tuottamaan haitallista koodia.

Seuraava askel ohjelmoijalle saatavilla olevien työkalujen sijaan on täysin itsenäisesti toimiva tekoälyagentti (AGI artificial general intelligence). Tämän ennustetaan olevan seuraava kehitysaskel nyt saatavilla olevista menetelmistä, kun kielimallien kehitys on saavuttanut huippunsa. Lyhenne AGI tulee englannin kielen termistä "artificial general intelligence" ja sillä tarkoitetaan tällä hetkellä vielä hypoteettista konseptia, jossa tekoälyn ajama agentti pystyisi suoriutumaan kaikista ihmisten tai eläimien älykkyyttä vaativista tehtävistä näitä paremmin ja tehokkaammin. (OpenAI 2023.) AGI:n saapumista yleiseen käyttöön ei voida vielä tarkasti ennustaa, ja sen vaatimasta kehitysajasta on monta erilaista mielipidettä. Tällä hetkellä arviot vaihtelevat muutamista vuosista kymmeneen vuosiin.



Yhteenvetona voitaisiin siis sanoa, että tekoälyn kehitys tuo mukanaan paljon hyvää ja paljon haasteita. Se on jo nyt muuttanut monia elämän osa-alueita, ja moni niistä tulee vielä muuttumaan huomattavasti. Tekoälyn käytössä täytyy olla varovainen sekä harkitsevainen. Lainsäädäntö ja ihmisen ohjaus tai valvonta tulee varmasti kehittymään hieman tekoälyn kehityksestä jäljessä, ja hyvän tasapainon ja yhteistyön löytäminen tulee viemään aikaa. Nähtäväksi jää, mitä kaikkea seuraavien kymmenien vuosien aikana tapahtuu.

## Lähteet

Alzubaidi, L., Zhang, J., Humaidi, A.J. ym. 2021. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Viitattu 3.5.2023. <https://doi.org/10.1186/s40537-021-00444-8>

Auto-GPT 2023. An Autonomous GPT-4 Experiment. Viitattu 11.5.2023. <https://github.com/Significant-Gravitas/Auto-GPT>

B.J. Copeland 2023. Artificial Intelligence. Viitattu 3.5.2023. <https://www.britannica.com/technology/artificial-intelligence>

Brown ym. 2020. Language Models are Few-Shot Learners. Viitattu 2.5.2023. <https://arxiv.org/pdf/2005.14165.pdf>

Dartmouth College 2023. Artificial Intelligence Coined at Dartmouth. Viitattu 2.5.2023. <https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth>

IBM 2023a. What is machine learning. Viitattu 3.5.2023. <https://www.ibm.com/topics/machine-learning>

IBM 2023b. What is deep learning. Viitattu 3.5.2023. <https://www.ibm.com/topics/deep-learning#:~:text=the%20next%20step-,What%20is%20deep%20learning%3F,from%20large%20amounts%20of%20data>.

IBM 2023c. What are neural networks. Viitattu 3.5.2023. <https://www.ibm.com/topics/neural-networks>

OpenAI 2023. GPT-4 Technical Report. Viitattu 5.5.2023. <https://cdn.openai.com/papers/gpt-4.pdf>

