Lassi Ylirautalahti

Tekoälyavusteisen ohjelman käyttö ohjelmoinnissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

8.5.2023

Tiivistelmä

Tekijä:	Lassi Ylirautalahti
Otsikko:	Tekoälyavusteisen ohjelman käyttö ohjelmoinnissa
Sivumäärä:	25 sivua
Aika:	8.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaaja:	Lehtori Miikka Mäki-Uuro

Insinöörityön tavoitteena oli hahmottaa, millainen työkalu Codex Sandbox on ohjelmoinnissa. Työssä perehdyttiin myös siihen keskeisesti liittyviin käsitteisiin, kuten tekoäly ja sen osa-alueet, OpenAI, Codex ja GitHub Copilot.

Koneoppiminen on tekoälyn osa-alue, jonka avulla tietokoneet oppivat ja tekevät ennusteita datasta. Kielimallit, kuten OpenAI:n GPT-3, ovat tekoälyjärjestelmiä, jotka käsittelevät ja tuottavat ihmiskielen kaltaista tekstiä oppimalla malleista ja asiayhteyksistä. OpenAI:n Codex on kehittynyt kielimalli, joka käyttää koneoppimistekniikoita, kuten syviä neuroverkkoja, ymmärtääkseen ja luodakseen koodia eri ohjelmointikielillä.

Työssä toteutettiin pienimuotoinen tapaustutkimus, jolla testattiin Codex Sandboxin mahdollisuuksia auttaa ohjelmoinnissa. Sekä opinnäytetyö että tapaustutkimus toteutettiin itsenäisesti. Tapaustutkimuksessa luotiin mobiilialustalle pelillistetty askelmittari, joka toimii hyödyntäen puhelimen kiihtyvyysanturia, jolla saatiin laskettua käyttäjän askeleet. Käyttäjällä on valittavissa taitoja, joita voidaan kehittää kävelemällä.

Codexin Sandboxin käyttö kuitenkin osoittautui rajoitetuksi, eikä sillä toteutettu koodi vastannut oletuksia. Codex Sandbox sai toteutettua yksittäisiä toiminnallisuuksia, jotka toimivat itsekseen, mutta yhdistetty kokonaisuus ei ollut mahdollinen Codex Sandboxin kanssa. Codex Sandboxin käyttäjäkokemus kuitenkin osoittautui opettavaiseksi, ja sen käyttö herätti mielenkiinnon muita tekoälyavusteisia ratkaisuja kohtaan.

Avainsanat: tekoäly, OpenAI, Codex Sandbox

Abstract

Author:	Lassi Ylirautalahti
Title:	Usage of AI assisted program in programming
Number of Pages:	25 pages
Date:	8 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Mobile Solutions
Instructor:	Miikka Mäki-Uuro, Senior Lecturer

The aim of the thesis was to understand what kind of tool Codex Sandbox is in programming. The thesis also introduces concepts that are important to the work, such as AI and its components, OpenAI, Codex and GitHub Copilot.

Machine learning is a branch of artificial intelligence that enables computers to learn and make predictions from data. Language models, like OpenAI's GPT-3, are AI systems that process and generate human-like text by learning patterns and context. OpenAI Codex is an advanced language model that uses machine learning techniques, including deep neural networks, to understand and generate code in various programming languages.

A small-scale case study was conducted to test the potential of Codex Sandbox in programming. Both the thesis and the case study were conducted independently. In the case study, a gamified pedometer was created for mobile platform that works by using the accelerometer of a phone to calculate the user's steps. The user has a choice of skills that can be developed by walking.

However, the Codex Sandbox proved to be limited in its usage and the code implemented with it did not meet expectations. Codex Sandbox was able to implement individual functionalities that worked on their own, but a combined whole was not possible with Codex Sandbox. However, the Codex Sandbox user experience proved to be educational, and its use opened interest in other AI-assisted solutions.

Keywords: Artificial Intelligence, OpenAI, Codex Sandbox

Sisällys

1 Johdanto

Ohjelmointi ja tekoäly ovat nopeasti kehittyviä aloja, jotka muuttavat tapoja elää ja työskennellä. Yksi tekoälyn kehittämisen edelläkävijäyrityksistä on OpenAI, joka on luonut useita uraauurtavia teknologioita, kuten Codex AI -mallin ja GitHub Copilotin.

Insinöörityössä perehdytään Codex Sandboxin eli Codex-testiympäristön ominaisuuksiin ja tutkitaan, miten sitä voidaan käyttää helpottamaan ohjelmointitehtäviä. Tästä eteenpäin Codex Sandboxiin viitataan Codex-testiympäristönä, ellei toisin mainita. Tutkitaan myös hieman mahdollisia haasteita ja eettisiä kysymyksiä, jotka liittyvät sekä Codexin että Copilotin käyttöön ohjelmistokehityksessä.

Codexin käytännön soveltamisen havainnollistamiseksi tehdään tapaustutkimus, jossa käytetään Codex-testiympäristöä luomaan pelillistetty askelmittarisovellus mobiilialustalle. Hyödyntämällä Codexin tehokkaita tekoälyominaisuuksia pyritään luomaan toimiva sovellus, joka osoittaa tekoälyn mahdollisuudet ohjelmistokehityksessä.

Tämän tapaustutkimuksen avulla tarkastellaan Codexin tehokkuutta ja rajoituksia reaalimaailman sovelluksen kehittämisessä. Käydään myös läpi sen haasteita ja mahdollisia tekijänoikeuskysymyksiä, joita voi syntyä käytettäessä Codexin ja Copilotin kaltaisia tekoälymalleja ohjelmistokehityksessä.

Insinöörityön tavoitteena on lisätä ymmärrystä tekoälyn mahdollisuuksista ja rajoituksista ohjelmistokehityksessä ja antaa tietoa mahdollisista haasteista ja eettisistä näkökohdista, jotka on otettava huomioon näiden teknologioiden kehittyessä edelleen.

2 Avainkäsitteet

Ohjelmointi, tekoäly, koneoppiminen, syväoppiminen ja kielimallit ovat kaikki toisiinsa liittyviä käsitteitä, jotka ovat mullistaneet tietojenkäsittelytieteen alan viime vuosina. Ohjelmoinnilla tarkoitetaan koodin kirjoittamista tietokoneen ohjeistamiseksi suorittamaan jokin tehtävä, kun taas tekoälyllä tarkoitetaan koneiden kykyä suorittaa tehtäviä, jotka yleensä vaativat ihmisille ominaisia kognitiivisia taitoja, kuten puheentunnistusta tai kuva-analyysiä (Schroer 2023).

Tekoälyä on myös erityyppistä, kuten heikko ja vahva tekoäly. Heikko tekoäly viittaa tekoälyjärjestelmiin, jotka on suunniteltu suorittamaan tiettyjä tehtäviä, kun taas vahva tekoäly, joka tunnetaan myös nimellä yleinen tekoäly, viittaa tekoälyjärjestelmiin, jotka pystyvät suorittamaan mitä tahansa älyllistä tehtävää, johon ihminen pystyy. Nykyhetkellä yleinen tekoäly on kuitenkin vielä teoreettisella tasolla. (Frankenfield 2023.)

Koneoppiminen on tekoälyn osa-alue, johon kuuluu algoritmeja, joiden avulla koneet voivat oppia datasta malleja ja yhtäläisyyksiä (Frankenfield 2023). Syväoppiminen on koneoppimisen alaryhmä, joka perustuu keinotekoisiin neuroverkkoihin, jotka mahdollistavat monimutkaisempien mallien oppimisen (What is deep learning). Kielimallit ovat tekoälymalleja, jotka on erityisesti suunniteltu käsittelemään ja ymmärtämään ihmisen kieltä, mikä tekee niistä olennaisesti tärkeän työkalun esimerkiksi luonnollisen kielen käsittelyssä ja tekstin tuottamisessa (Voita 2023). Tässä luvussa tarkastellaan kutakin näistä keskeisistä käsitteistä.

2.1 Ohjelmointi

Ohjelmointi on prosessi, jossa ihmisen luettavissa olevia ohjeita ja logiikkaa kirjoitetaan koneellisesti toteutettaviksi käskyiksi, joita tietokone ymmärtää ja pystyy toteuttamaan. Ohjelmoinnin yleisimpänä osana on koodin kirjoittaminen valitulla ohjelmointikielellä, johon kuuluu spesifinen syntaksi sekä kokoelma ohjeita,

joita tietokone ymmärtää. Ohjelmoinnilla voidaan viitata mihin tahansa ohjelmiston luontiprosessin kohtaan, johon sisältyy koodin kirjoittaminen, testaus sekä virheenkorjaus. Koodaus on olennainen osa ohjelmointia ja ohjelmistokehitystä. Se vaatii vahvan ymmärryksen tietotekniikan konsepteista, algoritmeista ja tietorakenteista. Ohjelmointia käytetään ohjelmistosovellusten, verkkosivujen, käyttöjärjestelmien, mobiilisovellusten ja muiden tietokoneohjelmien luomiseen. Kuten ohjelmoinnissa, myös koodauksessa tarvitaan loogista ajattelua, tarkkuutta ja ongelmanratkaisutaitoja. (Perez-Gascon & Weinstein 2022.)

2.2 Tekoäly

Tekoälyllä tarkoitetaan ihmisille olennaisen älykkyyden simulointia koneissa. Tekoäly on ohjelmoitu toteuttamaan tehtäviä, jotka tyypillisesti vaativat ihmisen kognitiota, kuten oppimista, ongelmanratkaisua, päätöksentekoa, havaintokykyä ja kielen tunnistamista. (Schroer 2023.)

Tekoälyjärjestelmät käyttävät algoritmeja ja matemaattisia malleja analysoidakseen dataa, tunnistaakseen säännönmukaisuuksia ja tehdäkseen ennusteita tai päätöksiä dataan perustuen. Tekoälyä käytetään useissa erilaisissa sovelluksissa, kuten luonnollisen kielen prosessoinnissa, kuvan tunnistuksessa, robotiikassa, puheen tunnistuksessa ja useissa muissa. Sen potentiaalinen vaikutus yhteiskuntaan ja teollisuuteen on laaja ja nopeasti kasvava. (Frankenfield 2023.)

2.2.1 Heikko ja vahva tekoäly

Heikolla tekoälyllä, joka tunnetaan myös nimellä kapea tekoäly, tarkoitetaan tekoälyä, joka on suunniteltu automatisoimaan ja suorittamaan sille määrätty tehtävä tai tehtäväkokonaisuus. Se keskittyy kehittämään algoritmeja, jotka suorittavat määrätyn funktion, kuten kuvantunnistus, puheentunnistus tai pelin pelaaminen, mahdollisimman täydellisesti, joissain tapauksissa jopa paremmin kuin ihmiset. Heikko tekoäly on rajoitettu sille suunniteltuun tehtävään, eikä sillä ole yleistä älykkyyttä tai tietoisuutta. (Glover 2022.)

Vahvalla tekoälyllä viitataan tekoälyyn, jolla on ihmisen kaltainen älykkyys ja tietoisuus. Vahva tekoäly on vielä teoreettinen konsepti, eikä vielä ole luotu yhtään tietokoneohjelmaa, jota voitaisiin pitää vahvana tekoälynä. Vahva tekoäly pystyisi ymmärtämään, päättelemään ja oppimaan samalla tavalla kuin ihminen. Vahva tekoäly ei olisi rajoittunut tiettyyn tehtävään, vaan se pystyisi soveltamaan älyään laajaan valikoimaan ongelmia ja tilanteita. (Glover 2022.)

Sekä heikko että vahva tekoäly ovat tärkeitä tutkimusalueita tekoälyn saralla, ja molemmilla on erilaiset implikaatiot tulevaisuuden teknologiaan ja yhteiskuntaan. Heikkoa tekoälyä käytetään jo moniin tarkoituksiin, kuten hakukoneissa, digiavustajissa ja suosittelujärjestelmissä. Jos vahva tekoäly saavutettaisiin, se voisi vaikuttaa merkittävästi talouteen, yhteiskuntaan ja ihmisten perusolemuksiin. (Glover 2022.)

2.2.2 Koneoppiminen ja syväoppiminen

Koneoppiminen on tekoälyn osa-alue, johon kuuluu sellaisten algoritmien kehittäminen, jotka voivat automaattisesti oppia malleja ja yhteyksiä datassa. Algoritmit on suunniteltu siten, että ne parantavat omaa tehokkuuttaan ajan kanssa käymällä uutta dataa läpi ja muuttamalla parametrejään asianmukaisesti, täten imitoiden ihmisten tapaa oppia. Koneoppimisen algoritmit voivat olla valvottuja, valvomattomia tai osittain valvottuja, ja niitä voidaan käyttää useisiin eri tarkoituksiin, kuten kuvien ja puheen tunnistamiseen tai ennakoivaan analyysiin. (What is machine learning.)

Valvotussa oppimisessa jokaisella koulutusdatan esimerkillä on syöttötieto ja tulostieto. Valvotun oppimisen tavoitteena on, että malli oppii ennustamaan oikean tuloksen, kun se saa uuden syötteen, jota se ei ole nähnyt aiemmin. Koulutusprosessissa mallin parametreja säädetään siten, että mallin ennustetun tuotoksen ja todellisen tuotoksen välinen ero minimoidaan, kuten määritetyssä tulostietokokonaisuudessa on määritelty. (What is machine learning.)

Esimerkkinä voidaan ottaa malli, joka on koulutettu ennustamaan sähköpostin tekstin perusteella, onko sähköposti roskapostia vai ei. Tässä tapauksessa merkitty tietokokonaisuus koostuisi sähköposteista, jotka on merkitty joko "roskapostiksi" tai "ei roskapostiksi". (Thorn 2020.)

Valvomattomassa oppimisessa koneoppimismalli koulutetaan merkitsemättömällä tietokokonaisuudella, jossa ei ole tulostunnisteita. Valvomattoman oppimisen tavoitteena on löytää datasta malleja tai rakenteita, joita voidaan käyttää samankaltaisten esimerkkien ryhmittelyyn tai datan poikkeamien pienentämiseen. Koulutusprosessiin kuuluu mallin parametrien säätäminen tietyn kohdefunktion optimoimiseksi. (Thorn 2020.)

Esimerkkinä voidaan ottaa malli, joka on koulutettu ryhmittämään asiakkaiden käyttäytymistä koskevat tiedot eri segmentteihin, jotta asiakkaiden käyttäytymismalleja voidaan ymmärtää paremmin. Tässä tapauksessa ei ole lähtömerkintöjä ja tavoitteena on löytää datasta rakenne, jonka avulla voidaan ryhmitellä samankaltaiset käyttäytymismallit yhteen. (Thorn 2020.)

Vahvistusoppimisessa malli koulutetaan tekemään useita päätöksiä valitussa ympäristössä ja tavoitteena on maksimoida kumulatiivinen palkintosignaali. Malli oppii kokeilemalla ja erehtymällä, ja se saa tekemiensä päätösten perusteella palautetta palkkioina tai rangaistuksina. Harjoitteluprosessissa mallin parametreja säädetään, jotta se oppii toimintatavan, joka maksimoi odotetun kumulatiivisen palkkion ajan myötä. (Thorn 2020.)

Esimerkkinä voidaan ottaa malli, joka on koulutettu pelaamaan shakkipeliä. Malli oppii tekemään siirtoja laudan senhetkisen tilan ja pelin voittamistavoitteen perusteella. Se saisi palautetta palkintosignaalina (pelin voittaminen) tai rangaistussignaalina (pelin häviäminen) jokaisen siirron jälkeen ja sovittaisi toimintatapaansa sen mukaisesti maksimoidakseen kumulatiivisen palkkionsa pelin aikana. (Thorn 2020.)

Syväoppiminen on koneoppimisen osa-alue, joka perustuu keinotekoisiin neuroverkkoihin. Algoritmit voivat oppia tunnistamaan monimutkaisia kuvioita ja suhteita datassa, joten ne soveltuvat hyvin esimerkiksi kuvan- ja puheentunnistukseen, luonnollisen kielen käsittelyyn ja autonomiseen ajamiseen. Syväoppiminen tarvitsee suuren määrän dataa ja paljon laskentatehoa. Syväoppiminen on kasvattanut suosiotaan viime vuosina, koska sen avulla voidaan saavuttaa huippuluokan suorituskyky monissa sovelluksissa. (What is deep learning.)

Koneoppiminen on siis tekoälyn osa-alueista yleisin, ja se kattaa erilaisia algoritmeja ja tekniikoita, joilla voidaan automaattisesti oppia kuvioita ja suhteita datasta. Syväoppiminen on koneoppimisen erikoistunut osa-alue, jossa käytetään keinotekoisia neuroverkkoja monimutkaisten kuvioiden ja suhteiden oppimiseen datasta. Sekä koneoppiminen että syväoppiminen ovat tärkeitä tekoälyn tutkimusalueita, ja niillä on lukuisia käyttötarkoituksia monilla aloilla, kuten terveydenhuollossa, rahoituksessa, lainvalvonnassa ja liikenteessä. (What is deep learning.)

2.3 Kielimalli

Luonnollisen kielen käsittelyssä kielimalli on tilastollinen malli, jota käytetään kielen rakenteen ja yhtäläisyyksien analysointiin ja ymmärtämiseen. Se pystyy tähän analysoimalla suuria määriä tekstidataa ja käyttämällä tätä tietoa ennustamaan tietyn kielen sanajoukkojen todennäköisyysjakaumaa. (Lutkevich 2020.)

Kielimallit toimivat tarkastelemalla lauseiden sanojen välisiä malleja ja suhteita, kuten sitä, kuinka usein tietyt sanat esiintyvät yhdessä tai missä yhteydessä. Analysoimalla näitä malleja kielimalli voi tehdä ennusteita siitä, mitkä sanat tulevat todennäköisesti seuraavaksi tietyssä lauseessa. (Lutkevich 2020.)

Kielimallit voidaan kouluttaa erilaisilla tekstiaineistoilla, kirjoista ja artikkeleista aina sosiaalisen median viesteihin ja keskusteluihin. Mitä suuremmalla määrällä dataa kielimalli koulutetaan, sitä paremmin se pystyy ennustamaan kielen sanajoukkojen todennäköisyysjakaumaa. (Voita 2023.)

Kun kielimalli on koulutettu, sitä voidaan käyttää erilaisissa luonnollisen kielen prosessointitehtävissä, kuten puheentunnistuksessa, konekääntämisessä ja tekstin tuottamisessa. Kielimalleja käytetään usein rakennuspalikoina monimutkaisemmissa malleissa, kuten neuraalisissa konekäännösjärjestelmissä, joissa ne auttavat tuottamaan sujuvampia ja luonnollisemmalta kuulostavia tuotoksia. (Lutkevich 2020.)

3 OpenAI-organisaatio ja Codex-kielimalli

OpenAI on tutkimusorganisaatio, joka on tekoälyteknologian kehittämisen eturintamassa. Sen viimeaikaisiin läpimurtoihin kuuluu Codex, tekoälyjärjestelmä, joka pystyy luomaan koodia luonnollisen kielen kehotteilla. Codex on rakennettu GPT-arkkitehtuurin päälle, samaa arkkitehtuuria, jota käytetään OpenAI:n tunnetuissa kielimalleissa, kuten GPT-3:ssa. (Exploring OpenAI Codex: A Comprehensive Guide 2023.) Codex on integroitu GitHubin kehitysympäristöön Copilot-nimisenä työkaluna (Your AI pair programmer). Tässä luvussa tarkastellaan OpenAI:ta, Codexia ja Copilotia lähemmin.

3.1 OpenAI-organisaatio

OpenAI on tekoälyn tutkimusorganisaatio, joka koostuu tutkijoista, insinööreistä ja tiedemiehistä, jotka ovat keskittyneet kehittämään ja edistämään tekoälyä turvallisesti ja ihmiskuntaa hyödyttävällä tavalla. Organisaation perusti vuonna 2015 joukko teknologiateollisuuden tunnettuja henkilöitä, kuten Elon Musk, Sam Altman, Greg Brockman, Ilya Sutskever ja muut. (Verna 2023.)

OpenAI:n tehtävänä on kehittää ja edistää tekoälyä tavalla, joka hyödyttää koko ihmiskuntaa. Tämän saavuttamiseksi se tekee tutkimusta tekoälyn eri aloilla, kuten koneoppimisessa, luonnollisen kielen käsittelyssä, robotiikassa ja tietokonenäössä. (OpenAI Charter 2018.)

OpenAI on antanut merkittävän panoksen tekoälyn alalle perustamisestaan lähtien. Se on kehittänyt useita vaikutusvaltaisia tekoälymalleja, kuten GPT-3-kielimallin, jota on pidetty merkittävänä läpimurtona luonnollisen kielen käsittelyssä. Se tukee tutkimusta, joka koskee generatiivisen tekoälyn aiheuttamia erityisiä luottamus- ja turvallisuushaasteita, jotta se voi parantaa turvallisuutta ekosysteemiensä ulkopuolella. (OpenAI and ChatGPT: All you Want to Know 2023.)

GPT-mallit, eli suomeksi generoiva esikoulutettu transformer-malli (Mikä ihmeen GPT? – Koneoppimisinsinööri vastaa 2023), koulutetaan valvomattomalla oppimisella, mikä tarkoittaa, että ne oppivat ennustamaan seuraavan sanan tekstisarjassa analysoimalla suuria tietomääriä. Mallit koulutetaan etukäteen massiivisella tekstiaineistolla, minkä jälkeen niitä hienosäädetään tietyissä tehtävissä, kuten kielenkääntämisessä tai tekstin luokittelussa. (Wolfe 2022.)

Yksi GPT-mallien tärkeimmistä ominaisuuksista on niiden kyky tuottaa yhtenäistä ja realistista ihmisen kaltaista tekstiä. Malleja on käytetty erilaisiin sovelluksiin, kuten kielenkääntämiseen, kysymyksiin vastaamiseen ja jopa luovaan kirjoittamiseen. (Ali 2023.)

Tutkimustyön lisäksi OpenAI tekee yhteistyötä teollisuuskumppaneiden kanssa tuodakseen tekoälyteknologioita markkinoille ja edistääkseen tehtäväänsä kehittää tekoälyä turvallisella ja hyödyllisellä tavalla. OpenAI pyrkii myös valistamaan yleisöä tekoälyn mahdollisuuksista ja vastuullisen tekoälyn kehittämisen tärkeydestä. (Verna 2023.)

3.2 Codex-kielimalli

OpenAI:n Codex on tekoälyjärjestelmä, joka voi kirjoittaa koodia useilla ohjelmointikielillä ohjelmoijan luonnollisella kielellä antamien syötteiden perusteella. Kouluttaakseen Codexia OpenAI käytti massiivista dataa julkisesti saatavilla

olevasta koodista, mukaan lukien GitHubista ja muista lähteistä peräisin olevasta koodista. Malli koulutettiin useilla eri ohjelmointikielillä, kuten Pythonilla, JavaScriptillä ja Go-kielellä. (Code completion.)

OpenAI:n Codex on hyvä esimerkki siitä, miten kielimalleja käytetään ohjelmoinnissa. Codex on tekoälyjärjestelmä, joka voi luoda koodia luonnollisen kielen kehotusten perusteella. Analysoimalla tiettyä kehotusta Codex voi päätellä, mitä kehittäjä yrittää saavuttaa, ja tuottaa sitten koodia, joka saavuttaa halutun lopputuloksen. (Exploring OpenAI Codex: A Comprehensive Guide 2023.)

Yksi Codexin tärkeimmistä ominaisuuksista on sen kyky ymmärtää ohjelmointikieliä ja koodaussyntaksia. Tämä tarkoittaa, että se voi tuottaa koodia, joka on paitsi oikein myös hyvin jäsenneltyä ja luettavaa. Tämä on merkittävä parannus verrattuna perinteisiin koodigeneraattoreihin, jotka tuottavat usein vaikeasti luettavaa ja ymmärrettävää koodia. (Exploring OpenAI Codex: A Comprehensive Guide 2023.)

Codexin kaltaisten kielimallien hyödyistä huolimatta niiden mahdollisista vaikutuksista ohjelmoinnin ammattikuntaan ollaan myös huolissaan. Jotkut sanovat, että nämä järjestelmät voivat lopulta korvata ihmisohjelmoijat, kun taas toiset pitävät niitä arvokkaana välineenä koodauksen tehokkuuden ja vaikuttavuuden parantamisessa. (Exploring OpenAI Codex: A Comprehensive Guide 2023.)

Kielimallit ovat olennainen osa nykyaikaista tekoälyä, ja niiden käyttö ohjelmoinnissa laajenee nopeasti. Codex on hyvä esimerkki siitä, miten kielimallien avulla voidaan luoda laadukasta koodia nopeasti ja tehokkaasti, mutta niiden pitkän aikavälin vaikutuksesta ohjelmointialalle on myös huolta. Kuten minkä tahansa uuden teknologian kohdalla, on tärkeää pohtia huolellisesti sekä kielimallien että niiden ohjelmointisovellusten hyötyjä ja mahdollisia haittoja. (Ot 2022.)

3.3 Codex-testiympäristö

OpenAI:n Codex testiympäristö on työkalu, jonka avulla kehittäjät voivat testata ja ajaa JavaScript-koodia turvallisessa, eristetyssä ympäristössä. Testiympäristö on rakennettu OpenAI Codex AI:n päälle, joka käyttää koneoppimista koodin tuottamiseen luonnollisen kielen kuvausten perusteella. (Code completion.)

Testiympäristö toimii seuraavasti:

• Käyttäjät syöttävät JavaScript-koodia testiympäristöön. Koodi voidaan kirjoittaa manuaalisesti tai luoda Codex API:n avulla luonnollisen kielen kuvauksen perusteella. (Code completion.)

• Tämän jälkeen testiympäristö suorittaa koodin eristetyssä ympäristössä, mikä tarkoittaa, että sillä ei ole pääsyä ulkoisiin resursseihin tai kirjastoihin. Näin voidaan varmistaa, että koodia on turvallista ajaa eikä se aiheuta haittaa. (Code completion.)

• Kun koodia suoritetaan, testiympäristö seuraa kaikkia sen ympäristöön tekemiä muutoksia, kuten uusien muuttujien luomista tai olemassa olevien muuttujien muuttamista (Code completion).

• Kun koodi on suoritettu loppuun, testiympäristö palauttaa ympäristön lopullisen tilan käyttäjälle. Tämä sisältää kaikki koodin tuottamat tulosteet ja kaikki muuttujiin tai objekteihin tehdyt muutokset. (Code completion.)

Codex-testiympäristö on tehokas työkalu kehittäjille, jotka haluavat kokeilla koodia turvallisessa ja valvotussa ympäristössä. Sen avulla he voivat testata koodinpätkiä ja luoda koodia luonnollisen kielen kuvausten perusteella. (Exploring OpenAI Codex: A Comprehensive Guide 2023.)

3.4 GitHub Copilot-työkalu

GitHub Copilot on tekoälyavusteinen koodiavustaja, jonka GitHub on kehittänyt yhteistyössä OpenAI:n kanssa. Se käyttää OpenAI:n Codexia, neuroverkkomallia, joka on koulutettu massiivisella koodiaineistolla antamaan ehdotuksia koodin täydentämiseen ja luomiseen. (Your AI pair programmer.)

Kun käyttäjä kirjoittaa koodia integroidussa kehitysympäristössään, GitHub Copilot analysoi koodin kontekstin ja antaa ehdotuksia seuraavista koodiriveistä. Ehdotukset luodaan käyttäjän jo kirjoittaman koodin mallien ja rakenteiden sekä koodikannan laajemman kontekstin, kuten käytettyjen kirjastojen ja kehysten, perusteella. (Your AI pair programmer.)

GitHub Copilot luottaa ehdotusten antamisessa Codexin tehokkaisiin luonnollisen kielen käsittelyominaisuuksiin, joiden avulla se ymmärtää koodin merkityksen ja ohjelmoijan tarkoituksen. Näin GitHub Copilot pystyy tuottamaan laadukasta koodia, joka on syntaktisesti oikein ja vastaa käyttäjän toivomaa lopputulosta. (Your AI pair programmer.)

GitHub Copilot on suunniteltu oppimaan käyttäjän koodaustyylistä ja mukauttamaan ehdotuksiaan sen mukaisesti. Kun käyttäjä jatkaa työskentelyä työkalun kanssa, GitHub Copilotista tulee entistä tarkempi ja tehokkaampi ja se antaa entistä yksilöllisempiä ja tarkoituksenmukaisempia ehdotuksia. (Your AI pair programmer.)

GitHub Copilot, kuten mikä tahansa tekoälytyökalu, joka tuottaa koodia, herättää huolta mahdollisista tekijänoikeusongelmista. Koska työkalu ehdottaa koodinpätkiä, jotka perustuvat koodiin, johon se on koulutettu, on olemassa riski, että se voi tuottaa koodia, joka loukkaa jonkun tekijänoikeuksia. GitHub on kuitenkin ilmoittanut, että se on ryhtynyt toimenpiteisiin tekijänoikeusrikkomusten riskin pienentämiseksi kouluttamalla Copilotia julkisesti saatavilla olevalla ja luvallisesti lisensoidulla koodilla sekä ottamalla käyttöön koodianalyysijärjestelmän, joka tarkastaa mahdolliset tekijänoikeuskysymykset luodussa koodissa.

Näistä toimista huolimatta on edelleen mahdollista, että Copilot tuottaa koodia, joka loukkaa jonkun tekijänoikeuksia, joten kehittäjien on tärkeää olla tietoisia näistä riskeistä ja ryhtyä asianmukaisiin toimiin niiden minimoimiseksi. (Jain 2022.)

4 Pelillistetty askelmittari tekoälyn avulla

Tässä luvussa selostetaan insinöörityössä tehdyn tapaustutkimuksen taustoja, kulkua ja tuloksia. Tapaustutkimuksen tarkoituksena oli havainnollistaa, miten OpenAI:n luoman Codex-testiympäristön avulla voidaan luoda pelillistetty askelmittari. Alussa käydään läpi, mikä tapaustutkimus on ja miten sitä voidaan käyttää ohjelmoinnin yhteydessä.

4.1 Tapaustutkimus tutkimusmenetelmänä

Tapaustutkimus on tutkimusmenetelmä, jossa tutkitaan ja analysoidaan syvällisesti tiettyä tapausta, ilmiötä tai tilannetta. Tapaustutkimuksessa tutkija kerää yksityiskohtaista tietoa tutkimuskohteesta käyttämällä erilaisia menetelmiä, kuten haastatteluja, kyselyjä, havainnointia ja asiakirjojen analysointia. Tavoitteena on saada syvällinen ymmärrys tutkimuskohteesta ja sen asiayhteydestä sekä tunnistaa malleja, suuntauksia ja oivalluksia, joita voidaan soveltaa samankaltaisiin tilanteisiin. (Runeson & Höst 2008.)

Koodaushankkeessa tapaustutkimusta voidaan käyttää usealla eri tavalla. Siinä voidaan esimerkiksi tutkia tietyn ohjelmointitekniikan tai algoritmin tehokkuutta todellisessa kontekstissa. Tutkija valitsee tietyn ohjelmistojärjestelmän tai sovelluksen ja kerää tietoja sen suorituskyvystä, käyttäjäpalautteesta ja muista merkityksellisistä tekijöistä. Analyysissä vertaillaan eri ohjelmointitekniikoiden suorituskykyä, tunnistetaan parannusalueita ja annetaan suosituksia tulevaa kehitystä varten. Toinen tapa, jolla tapaustutkimusta voitaisiin käyttää koodaushankkeessa, on tutkia tietyn teknologian tai työkalun käyttöä tietyllä alalla. (Runeson & Höst 2008.)

Tapaustutkimus voi olla arvokas tutkimusmenetelmä koodaushankkeessa, sillä sen avulla voidaan saada yksityiskohtaista tietoa tietystä asiayhteydestä tai ongelmasta ja tunnistaa malleja ja suuntauksia, jotka eivät ehkä näy laajemmasta näkökulmasta. Käyttämällä tapaustutkimusmenetelmää tutkijat voivat kerätä runsaasti ja yksityiskohtaista tietoa, jota voidaan hyödyntää, kun kehitetään uusia teknologioita, työkaluja ja lähestymistapoja, jotka on räätälöity tiettyihin yhteyksiin tai aloihin. (Runeson & Höst 2008.)

4.2 Pelillistetty askelmittari

Insinöörityössä toteutettiin pienimuotoinen tapaustutkimus, jossa hyödynnettiin OpenAI:n Codex-testiympäristöä. Tapaustutkimus toteutettiin helmikuussa 2023, ja sen tavoitteena oli luoda pelillistetty askellaskuri mobiilialustalle. Sovelluksen ominaisuuksiksi haluttiin askeleiden laskeminen ja tasot, joita käyttäjä kasvattaa kävellessään ja joista yksi voi olla valittuna kerrallaan. Lähes kaikki koodi toteutettiin Codexilla pieniä korjauksia lukuun ottamatta. Codex-testiympäristöä käytettiin Codex API:n sijaan tutkittaessa, millainen käyttäjäkokemus saadaan ja kuinka paljon voidaan saada aikaan käyttäen käyttäjäystävällistä koodin toteutusta, joka toimii luonnollisen kielen syötteillä. Mahdolliset muutokset toteutettiin Visual Studio Code-ohjelmalla. Alustavasti oletettiin, että Codex voisi toteuttaa projektin ilman suurempia ongelmia, mutta Codexin toteutukset eivät kuitenkaan aina vastanneet odotuksia. Tavoitteena oli hahmottaa, millaista hyötyä Codexin testiympäristöstä voisi olla ohjelmoinnissa.

Ohjelmointialustaksi valittiin React, ja ohjelmointikielenä päätettiin käyttää ReactJS:ää. Myöhemmin kuitenkin ilmeni, että ReactJS oli hieman hankala, sillä vaikka Codex tukee yli 12:ta ohjelmointikieltä, se ei osannut käyttää ReactJS:ää kovin hyvin eikä pystynyt näyttämään esikatselua toteutetusta koodista, kuten se pystyi esimerkiksi JavaScriptin kanssa.

Alussa Codexille tehtiin pyyntö luoda yleinen aloitussivu, jolle käyttäjä saapuu heti sovelluksen avautuessa. Tekoälylle ei annettu kovin tarkkoja ohjeita, sillä

haluttiin alustavasti nähdä, millaista jälkeä Codex saa aikaan. Aluksi etusivu luotiin käyttäen "React.createClass"-metodia, kuten esimerkkikoodissa 1 nähdään. Kuitenkin myöhemmin päätettiin käyttää ES6-luokkatyyliä, jossa luokat luodaan käyttämällä "Luokka extends React.Component" -tapaa. Tämä onnistui pyytämällä Codexia käyttämään haluttua tyyliä.

```
import React, { useState, useEffect } from 'react';

var React = require('react');
var ReactDOM = require('react-dom');
var HelloWorld = React.createClass({
  render: function() {
    return (
      <div>
        Hello World
      </div>
    );
  }
});
ReactDOM.render(
  <HelloWorld />,
  document.getElementById('container')
);
```

Esimerkkikoodi 1. Ensimmäinen Codexin tuotos annetun luonnollisen kielen kehotteen mukaan.

Toisessa osiossa pyydettiin Codexia luomaan askellaskuri käyttäen puhelimen kiihtyvyysanturia. Tähänkin osioon annettiin alustavasti erittäin avoin pyyntö, jotta voitiin nähdä, millä tavalla Codex toteuttaisi askellaskurin ilman tarkempia ohjeita. Codex ei kuitenkaan osannut luoda suoraan toimivaa kokonaisuutta, sillä se ei alkuun käyttänyt nuolifunktioita. Pyynnön antaminen Codexille siitä,

että haluttiin sen käyttävän nuolifunktioita, mahdollisti toimivamman ratkaisun luomisen, kuten esimerkkikoodissa 2 nähdään.

```
import React, { useState, useEffect } from 'react';

export default function SetpCalculation() {
  const [steps, setSteps] = useState(0);
  useEffect(() => {
    window.addEventListener('devicemotion', (event) => {
      const { accelerationIncludingGravity } = event;
      const { x, y, z } = accelerationIncludingGravity;
      const acceleration = Math.sqrt(x * x + y * y + z *
z);
      if (acceleration > 25 && z < 0) {
        setSteps(steps + 1);
      }
    });
  }, [steps]);
  return <div style={{ textAlign: 'center' }}>Steps:
{steps}</div>;
};
```

Esimerkkikoodi 2. Koodi laskee puhelimen kiihtyvyyden ja sen perusteella lisää askeleita. Askeleiden laskeminen ei kuitenkaan ota huomioon kuin puhelimen kiihtyvyyden, joten sen tarkkuus ei ole riittävä.

Kolmannessa osassa haluttiin, että tekoäly luo pelaajalle tasojärjestelmän, joka lähtee tasosta 1 ja korkein saavutettava taso on 99. Lisäksi annettiin alustavasti pyyntö painikkeesta, joka antaisi satunnaisen määrän kokemuspisteitä 0,1:n ja 1:n väliltä. Tämä oli alustavaa testausta varten, jonka oli tarkoitus olla väliaikaista. Tarvittavien kokemuspisteiden luominen jokaiselle tasolle onnistui hyvin,

mutta tasojen ja kokemuspisteiden laskeminen ja nostaminen ei aluksi onnistunut. Codex olisi lähtenyt luomaan 99 eri funktiota sekä kokemuspisteille että tasoille, kuten esimerkkikoodissa 3 näkyy.

```
function getXpToNextLevel() {
  return xpToLevel[level + 1] - xpToLevel[level];
}
function getXpToLevel(level) {
  return xpToLevel[level];
}
function getLevelFromXp(xp) {
  var level = 1;
  while (xp >= xpToLevel[level]) {
    level++;
  }
  return level - 1;
}
function getXpToLevelFromXp(xp) {
  return xpToLevel[getLevelFromXp(xp)];
}
function getXpToNextLevelFromXp(xp) {
  return xpToLevel[getLevelFromXp(xp) + 1] -
xpToLevel[getLevelFromXp(xp)];
}
function getXpFromLevel(level) {
  return xpToLevel[level];
}
function getXpFromLevelAndXpToLevel(level, xpToLevel) {
  return xpToLevel[level] + xpToLevel;
}…
```

Esimerkkikoodi 3. Codexin luoma tapa kasvattaa käyttäjän tasoa.

Tässä vaiheessa kuitenkin huomattiin, että Codexilla oli vaikeuksia vaihtaa tapaa, jolla tietty osio oli aiemmin luotu. Samaa tai samankaltaista koodia syntyi usein, vaikka Codexia pyydettiin luomaan koodia eri tavalla tai välttämään tiettyjä tapoja, joita se oli aikaisemmin käyttänyt.

FOR-silmukan tai IF-valintarakenteen luomista, jolla tason kasvatus voitiin toteuttaa, pyydettiin Codexilta monella eri tavalla, mutta se ei kuitenkaan pystynyt vaihtamaan tapaa, jolla se oli aiemmin tämän osion luonut. Lopulta päädyttiin luomaan yksinkertainen IF-valintarakenne, jolla kokemuspisteitä ja tasoja voitiin nostaa, kuten esimerkkikoodissa 4 nähdään.

```
levelUpCalc = (xp) => {
    if (xp >= this.xpToLevel[this.state.level]) {
        this.setState({
            level: this.state.level + 1
        });
    }}

    handleClick = () => {
        var xp = this.state.xp;
        var xpToAdd = Math.random() * (1 - 0.1) + 0.1;
        xp += xpToAdd;
        this.setState({
          xp: Math.round(xp * 100) / 100
        })
        this.levelUpCalc(this.state.xp);
    };
```

Esimerkkikoodi 4. Lopullinen kokemuspisteiden ja tasojen nosto.

Viimeiseksi pyydettiin tekoälyä luomaan erilaisia taitoja, joita käyttäjä voi kehittää. Codexia pyydettiin luomaan viisi eri taitoa, joilla on nimi, kuva ja kuvaus ja joista vain yksi voi olla valittuna kerrallaan. Lisäksi valitun taidon taustan tulee

muuttua vihreäksi. Eri taitojen luominen sujui Codexilta hyvin, mutta kehitettäväksi halutun taidon valitseminen ei onnistunut, mikä esimerkkikoodissa 5.

```
var Skills = React.createClass({
  getInitialState: function() {
    return {
      selectedSkill: ''
    };
  },
  handleClick: function(name) {
    this.setState({
      selectedSkill: name
    });
  },
  render: function() {
    var skills = this.props.skills.map(function(skill) {
      return (
        <Skill name={skill.name} description={skill.de-
scription} image={skill.image} onClick={this.handleClick}
/>
      );
    }.bind(this));
    return (
      <div>
        {skills}
      </div>
    );
  }
});
```

Koodi 5. Codexin luoma tapa vaihtaa valittua taitoa.

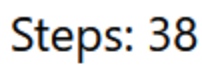
Codexille annettiin useita eri tavalla sanoitettuja pyyntöjä kokeilun vuoksi, sillä haluttiin varmistaa, olivatko aikaisemmat pyynnöt olleet epäselviä. Tämä ei kuitenkaan auttanut, sillä Codexin toteutuksessa jokainen taito voitiin valita samaan aikaan, jolloin aiemmin valitulta taidolta ei poistettu vihreää taustaa. Lopulta valinnan tekeminen päädyttiin korjaamaan käsin, mikä osoittautui kuitenkin erittäin helpoksi nopean tarkastelun jälkeen, koska tarvittiin vain yksinkertainen setState-silmukka kohtaan, jossa uusi taito valittiin.

Kun kaikki osat toimivat itsekseen, yritettiin Codexilla yhdistää kyseiset osiot, mutta se ei osoittautunut mahdolliseksi ainakaan Codex-testiympäristössä, sillä jo valmiita osioita ei voitu liittää siihen, eikä se voinut ottaa yhteyttä GitHub-arkistoon, jossa koodi sijaitsi. Tämä osio olisi mahdollisesti pystytty toteuttamaan GitHubin Copilotilla, mutta se ei kuitenkaan kuulunut alkuperäiseen suunnitelmaan testata Codex-testiympäristön mahdollisuuksia ohjelmoinnissa.

4.3 Lopputulos

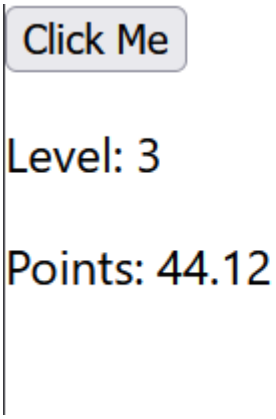
Tapaustutkimuksen ohjelmointiosion lopputuloksena syntyi yksittäisiä toiminnallisuuksia, jotka eivät toiminnoiltaan ole kovinkaan komplekseja. Vaikka Codexin tuottama kokonaisuus ei täysin vastannut odotuksia, nähtiin kuitenkin, mihin täysin luonnollisen kielen syötteisiin perustuva koodigenerointi pystyy.

Askellaskuriosio tuotti erittäin yksinkertaisen tavan laskea puhelimen liikkeen avulla askeleita (kuva 1). Osiosta jäi puuttumaan tarkempi tapa käyttää puhelimen kiihtyvyysanturia, jolloin useat virheellisesti lasketut askeleet jäisivät pois. Askellaskuri ei myöskään hyödynnä GPS-toiminnallisuutta, jolla saataisiin tarkempaa tulosta otetuista askeleista. Ulkonäöltään ei askeleiden laskeminen myöskään ole kovin kompleksi, sillä minkäänlaista muotoilua ei Codexilta pyydetty.



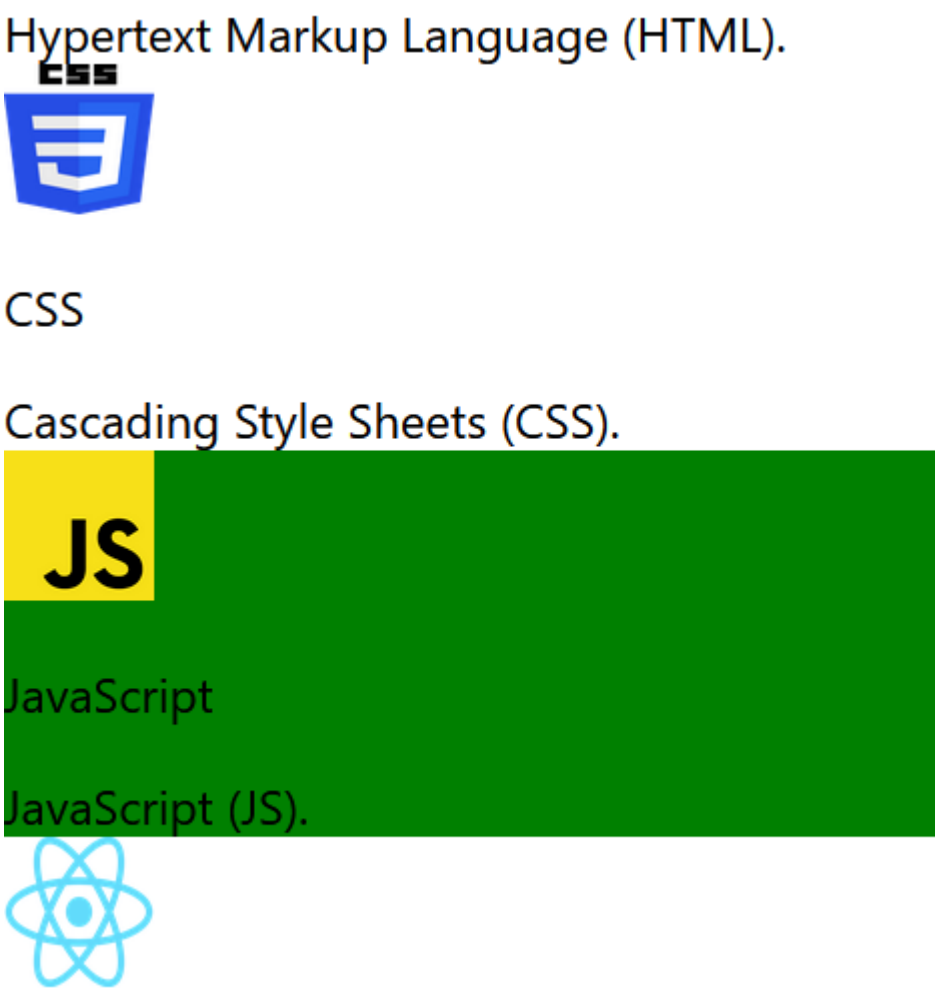
Kuva 1. Käyttäjän ottamien askeleiden näkymä.

Tasosysteemiosio oli Codexin osalta pienimmässä roolissa, sillä tässä osiossa Codex ei ymmärtänyt annettuja ohjeita ja tason nosto päädyttiin tekemään itse. Codexin osuudeksi jäi pelkästään tasojen kokemuspisteiden rajojen luominen, josta se kuitenkin suoriutui ongelmitta. Kokonaisuudessaan tasot ja niiden nosto oli kuitenkin helppo toteutus, josta ei jäänyt puuttumaan mitään (kuva 2).



Kuva 2. Painike, jolla käyttäjä voi lisätä satunnaisen määrän kokemuspisteitä itselleen, sekä käyttäjän taso ja käyttäjän saamat kokemuspisteet.

Taidot-osio koostui viidestä valittavasta taidosta, joista käyttäjä voi kerrallaan valita yhden (kuva 3). Ottamatta huomioon Codexin virhettä taidon valintaa tehdessä tämä osio sisälsi sille suunnitellun toiminnallisuuden, joten toteutus tehtiin onnistuneesti.



Kuva 3. Käyttäjän valittavissa olevat taidot, joista yksi voi olla valittuna kerrallaan.

Kokonaisuudessaan ohjelmointiosio ei vastannut oletuksia, vaan jäi suppeaksi. Yksittäisten toiminnallisuuksien yhdistäminen toimivaksi kokonaisuudeksi olisi ollut tärkeä osa tutkimusta, mutta se osoittautui mahdottomaksi Codexin puitteissa.

4.4 Huomiot

Insinöörityön tapaustutkimuksessa oli tavoitteena testata Codex-testiympäristön mahdollisuuksia koodaustyössä ja selvittää, kuinka hyvin tämä tekoäly avustaa ohjelmointia. Projektin aikana Codexia pyydettiin luomaan erilaisia koodinpätkiä

ja testattiin sen kykyä yhdistää luotuja osioita. Vaikka Codex osoitti hyvää osaamista monissa tehtävissä, oli sen käytössä myös haasteita. Esimerkiksi koodinpätkien yhdistäminen osoittautui mahdottomaksi eikä se pystynyt toteuttamaan joitain tehtyjä pyyntöjä, kuten taitojen valitsemista. Tämä johti siihen, että joitain ratkaisuja päädyttiin lopulta tekemään käsin.

Projekti antoi hyvän kuvan Codexin kyvyistä ja sen mahdollisuuksista avustaa ohjelmointia, mutta huomattiin kuitenkin, että se ei ole täydellinen ratkaisu ja että sen käytössä on edelleen haasteita. Tämä korostaa ihmisen roolia ohjelmoinnissa ja sitä, että ohjelmointiin tarvitaan edelleen ihmisen omaa osaamista ja päättelykykyä.

Maaliskuusta 2023 lähtien OpenAI on lopettanut Codex-mallin käytön, ennen kuin se edes pääsi ulos beetaversiosta. Se ja sen API eivät ole enää yleisön saatavilla. Tämä uutinen tuli julkiseen tietoon alle viikon varoitusajalla, joten monet Codexia käyttäneet kehittäjät jäivät tyytymättömiksi. Tämä vaikuttaa myös GitHub Copilotiin, joka on rakennettu Codexin varaan. Copilotin etusivulla todetaan edelleen, että Copilot toimii OpenAI Codexin avulla. On kuitenkin todennäköistä, että Copilot käyttää tulevaisuudessa OpenAI:n GPT-4-mallia. (Pandey 2023.)

5 Yhteenveto

Insinöörityössä tutkittiin Codex-testiympäristön hyödyntämistä ohjelmoinnissa. Tavoitteena oli kehittää mobiilialustalle sovellus, joka perustuu yksinomaan Codex-testiympäristön avulla toteutettuun koodiin. Lisäksi työssä käsiteltiin tarkemmin tekoälyä ja sen osa-alueita, jotka liittyvät olennaisesti aiheeseen. Tärkeimpänä näistä koneoppiminen ja kielimalli. Työn aikana tutustuttiin myös OpenAI:hin ja sen luomiin Codexiin ja GitHub Copilotiin.

Insinöörityö osoitti, että Codex-testiympäristön käyttäminen projektin luomiseen voi olla hyvä tapa tutustua ohjelmointiin sellaiselle, jolla ei ole kokemusta ohjelmoinnista. Vaikka insinöörityön projekti ei vastannut odotuksia lopputuotteen suhteen, Codex-testiympäristön käyttöprosessi tarjosi arvokkaita oppimiskokemuksia. Ensinnäkin Codex-testiympäristö oli käyttäjäystävällinen työkalu, joka tarjosi helppokäyttöisen käyttöliittymän ja pääsyn monenlaisiin ohjelmointikieliin ja -puitteisiin. Alusta tarjosi yksinkertaisen tavan luoda ja testata koodia. Lisäksi Codex-testiympäristö tarjosi uudenlaisen lähestymistavan projektien toteuttamiseen hyödyntämällä luonnollisen kielen syötteitä.

Insinöörityön lopputulos antoi näkemyksen tekoälyavusteiseen koodinluontiin Codex-testiympäristöä käyttäen. Työkalu on suunniteltu kokeiluun ja oppimiseen, eikä siinä välttämättä ole kaikkia vankemman koodausympäristön ominaisuuksia tai valmiuksia. Lisäksi Codex-testiympäristössä luodut projektit voivat olla laajuudeltaan tai toiminnallisuudeltaan rajallisia, mikä osoittautui todeksi useassa kohtaa tapaustutkimusta. Codex-testiympäristö voi kuitenkin auttaa ymmärtämään ohjelmoinnin perusteita ja antaa käsityksen siitä, millaista on työskennellä koodin kanssa. Vaikka projektista ei tulisikaan juuri sellainen kuin on toivonut, Codex-testiympäristön käyttökokemus voi olla ponnahduslauta kohti edistyneempiä koodausvälineitä ja -tekniikoita.

Codex-malli ja -testiympäristö eivät kuitenkaan ole enää käytettävissä, joten jatkossa tekoälyavusteista koodinluontia tulee kokeilla muilla olemassa olevilla malleilla, kuten työssä esitellyllä GitHub Copilotilla.

Lähteet

Ali, Fawad. 2023. GPT-1 to GPT-4: Each of OpenAI's GPT Models Explained and Compared. Verkkoaineisto. MakeUseOf. <https://www.makeuseof.com/gpt-models-explained-and-compared/>. Luettu 3.5.2023.

Code completion. Verkkoaineisto. OpenAI <https://platform.openai.com/docs/guides/code>. Luettu 12.2.2023.

Exploring OpenAI Codex: A Comprehensive Guide. 2023. Verkkoaineisto. Renaissance Rachel. <https://renaissancerachel.com/open-ai-codex/>. Luettu 3.5.2023.

Frankenfield, Jake. 2023. Artificial Intelligence: What It Is and How It Is Used. Verkkoaineisto. Investopedia. <https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp>. Luettu 25.4.2023.

Glover, E. 2022. Strong AI vs. Weak AI: What's the difference? Verkkoaineisto. Built In. <https://builtin.com/artificial-intelligence/strong-ai-weak-ai>. Luettu 4.3.2023.

Jain, Ayush. 2022. GitHub Copilot: The Latest in the List of AI Generative Models Facing Copyright Allegations. Verkkoaineisto. Analytics India Magazine. <https://analyticsindiamag.com/github-copilot-the-latest-in-the-list-of-ai-generative-models-facing-copyright-allegations/>. Luettu 2.5.2023.

Lutkevich, Ben. 2020. Language modeling. Verkkoaineisto. Tech Target. <https://www.techtarget.com/searchenterpriseai/definition/language-modeling>. Luettu 3.5.2023.

Mikä ihmeen GPT? – Koneoppimisinsinööri vastaa. 2023. Verkkoaineisto. a.i.mater. <https://aimater.com/mika-ihmeen-gpt-koneoppimisinsinoori-vastaa/>. Luettu 3.5.2023.

OpenAI and ChatGPT: All you Want to Know. 2023. Verkkoaineisto. Appmaster. <https://appmaster.io/blog/openai-and-chatgpt>. Luettu 4.3.2023.

OpenAI Charter. 2018. Verkkoaineisto. OpenAI <https://openai.com/charter>. Luettu 8.3.2023.

Ot, A. 2022. What Is OpenAI and Does It Really Make Coding Easier? Verkkoaineisto. MakeUseOf. <https://www.makeuseof.com/openai-coding/>. Luettu 04.3.2023.

Pandey, Mohit. 2023. OpenAI Might Invite Legal Trouble. Verkkoaineisto. Analytics India Magazine. <https://analyticsindiamag.com/openai-might-invite-legal-trouble/>. Luettu 2.5.2023.

Perez-Gascon, A. & Weinstein, J. 2022. What Is Coding? Verkkoaineisto. Career Karma. <https://careerkarma.com/blog/what-is-coding-used-for/>. Luettu 28.2.2023.

Runeson, H. & Höst, M. 2008. Guidelines for conducting and reporting case study research in software engineering. Verkkoaineisto. Springer Link. <https://link.springer.com/article/10.1007/s10664-008-9102-8>. Luettu 20.3.2023.

Schroer, A. 2023. What Is Artificial Intelligence? Verkkoaineisto. Built In. <https://builtin.com/artificial-intelligence>. Luettu 3.3.2023.

Thorn, James. 2020. The Good, the Bad, and the Ugly: Supervised, Unsupervised and Reinforcement Learning. Verkkoaineisto. Medium. <https://towardsdatascience.com/the-good-the-bad-and-the-ugly-supervised-unsupervised-and-reinforcement-learning-2ccf814c6bab>. Luettu 5.3.2023.

Verna, P. 2023. What to know about OpenAI, the company behind ChatGPT. Verkkoaineisto. The Washington Post. <https://www.washingtonpost.com/technology/2023/02/06/what-is-openai-chatgpt/>. Luettu 6.3.2023.

Voita, Lena. 2023. Language Modeling. Verkkoaineisto. <https://lena-voita.github.io/nlp_course/language_modeling.html>. Luettu 3.5.2023.

What is deep learning? Verkkoaineisto. IBM. <https://www.ibm.com/topics/deep-learning>. Luettu 5.3.2023.

What is machine learning? Verkkoaineisto. IBM. <https://www.ibm.com/topics/machine-learning >. Luettu 5.3.2023.

Wolfe, Cameron R. 2022. Language Models: GPT and GPT-2. Verkkoaineisto. Medium. <https://towardsdatascience.com/language-models-gpt-and-gpt-2-8bdb9867c50a>. Luettu 3.5.2023.

Your AI pair programmer. Verkkoaineisto. GitHub. <https://github.com/features/copilot>. Luettu 20.3.2023 & 02.05.2023.