

Janne Mäläskä

TEKOÄLYN KÄYTTÖ ANDROID-SOVELLUKSEN KEHITYKSESSÄ

TEKOÄLYN KÄYTTÖ ANDROID-SOVELLUKSEN KEHITYKSESSÄ

Janne Mäläskä
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä(t): Janne Mäläskä

Opinnäytetyön nimi: Tekoälyn käyttö Android-sovelluksen kehityksessä

Työn ohjaaja(t): Jukka Jauhiainen

Työn valmistusluku ja -vuosi: kevät 2024

Sivumäärä: 37

Tämä opinnäytetyö käsittelee tekoälyn hyödyntämistä ohjelmoinnin apuna. Työn tavoitteena oli selvittää, millaisia hyötyjä tekoälyyn perustuvien työkalujen käytöstä on, sekä tutkia tekoälytekniikoita, tekoälyn historiaa ja eettisiä kysymyksiä. Aihe valittiin henkilökohtaisesta kiinnostuksesta aiheita kohtaan.

Työssä tutkittiin myös Android-sovelluksen kehityksen periaatteita, työkaluja, käytäntöjä sekä tekoälyn hyödyntämistä osana Android-sovellusta. Opinnäytetyössä toteutettiin myös Android-sovellus, joka hyödynsi yleisimpiä tekoälytyökaluja ohjelmoinnissa.

Työn tulokset osoittavat, että tekoäly voi olla tehokas työkalu ohjelmoinnissa. Tekoäly voi auttaa ratkaisemaan ohjelmointiongelmia nopeammin kuin perinteiset tavat, mutta se ei kuitenkaan voi korvata ohjelmoijan omaa osaamista.

Asiasanat: tekoäly, ohjelmointi, Android

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author(s): Janne Mäläskä

Title of thesis: The Use of Artificial Intelligence in Android Software Development

Supervisor(s): Jukka Jauhiainen

Term and year when the thesis was submitted: Spring 2024

Number of pages: 37

This thesis deals with the use of artificial intelligence as a programming aid. The aim of the work is to find out about the benefits of using AI-based tools in programming, the history of AI and different AI technologies.

The work also examined the principles, tools, best practices, and the use of artificial intelligence in Android applications. The thesis also implemented an Android application that used the most common AI tools in programming.

The results of the work show that artificial intelligence can be an effective tool in programming, but it does not replace the skills of the programmer.

Keywords: artificial intelligence, programming, Android

SISÄLLYS

1	JOHDANTO	6
2	TEKOÄLYN PERUSTEET JA SOVELTAMINEN OHJELMOINNISSA.....	7
2.1	Tekoälyn historia ja perusteet.....	7
2.2	Tekoälytekniikat ohjelmoinnissa	7
2.3	Tekoälyn soveltaminen Android-sovellusten kehittämisessä.....	8
3	ANDROID-SOVELLUSTEN KEHITTÄMISEN PERIAATTEET, TYÖKALUT JA PARHAAT KÄYTÄNNÖT.....	10
3.1	Android-sovelluskehityksen periaatteet.....	10
3.2	Android-sovelluskehityksen työkalut.....	10
3.3	Parhaat käytännöt Android-sovelluskehityksessä	11
4	TEKOÄLYN HYÖDYNTÄMINEN ANDROID-SOVELLUKSISSA	13
4.1	Koneoppimisen integroiminen Android-sovelluksiin.....	13
4.2	Esimerkkejä tekoälypohjaisista Android-sovelluksista.....	13
4.3	Tekoälyn haasteet ja eettiset kysymykset	14
5	ANDROID-SOVELLUKSEN KEHITTÄMINEN KÄYTÄNNÖSSÄ	16
5.1	Sovelluksen ideointi.....	16
5.2	Sovelluksen suunnittelu.....	17
5.3	Sovelluksen ohjelmointi ja käytetyt tekoälytyökalut	18
5.3.1	GPT-4	18
5.3.2	ChatGPT ja Bing Chat.....	18
5.3.3	GitHub Copilot.....	22
5.3.4	Ohjelmointi käytännössä.....	24
5.3.5	Kehotteiden kehittäminen ja kehotemuotoilu.....	31
5.3.6	Ongelmatilanteet tekoälytyökalujen kanssa	31
6	POHDINTA.....	33
	LÄHTEET.....	34

1 JOHDANTO

Ohjelmoinnin osaamiselle on kysyntää monilla eri aloilla. Erityisesti mobiilisovellusten kehittäminen on ala, joka kasvaa ja kehittyy jatkuvasti. Android on yksi suosituimmista mobiilialustoista, ja sen sovellusten kehittäminen on tärkeä osa ohjelmointia. Tekoäly (Artificial Intelligence, AI) on noussut merkittäväksi teknologian kehityssuunnaksi, joka tarjoaa uusia mahdollisuuksia ohjelmointiin ja sovelluskehitykseen. Tässä opinnäytetyössä tarkastellaan tekoälyn hyödyntämistä ohjelmoinnin apuvälineenä Android-sovelluksen kehitysprosessissa.

Tämän opinnäytetyön tavoitteena on tutkia, kuinka tekoälyä voidaan hyödyntää esimerkiksi Android-sovellusten kehittämisessä ja millaisia etuja ja haasteita tekoälyn käyttö tuo mukanaan. Työssä käydään läpi erilaisia tekoälyyn perustuvia työkaluja ja menetelmiä, joita voidaan käyttää sovelluskehityksen eri vaiheissa, kuten ideoinnissa, suunnittelussa, ohjelmoinnissa, testauksessa ja julkaisussa.

Opinnäytetyö koostuu teoriaosuudesta ja käytännön osuudesta. Teoriaosuudessa käsitellään tekoälyn perusteita ja sen soveltamista ohjelmoinnissa sekä perehdytään Android-sovellusten kehittämisen periaatteisiin. Käytännön osuudessa toteutetaan esimerkkisovellus, jossa hyödynnetään tekoälyä eri vaiheissa. Sovelluksen kehitysprosessia ja sen tuloksia analysoidaan kriittisesti, jotta saadaan ymmärrystä tekoälyn vaikutuksesta sovelluksen laatuun ja kehitystyön tehokkuuteen.

Tämän opinnäytetyön tuloksien perusteella voidaan arvioida tekoälyn potentiaalia ohjelmoinnin apuvälineenä ja tunnistaa sen mahdollisuuksia sekä rajoitteita Android-sovellusten kehittämisessä. Työn tulokset voivat olla hyödyllisiä sekä ohjelmistokehittäjille, jotka haluavat hyödyntää tekoälyä työssään, että yrityksille ja organisaatioille, jotka pohtivat tekoälyn käyttöönottoa ohjelmistokehityksen osana.

Tämän opinnäytetyön avulla pyritään lisäämään ymmärrystä tekoälyn soveltamisesta ohjelmoinnin apuna ja tarjoamaan konkreettisia esimerkkejä sen hyödyntämisestä Android-sovellusten kehittämisessä. Työn tulokset voivat toimia pohjana jatkotutkimuksille ja auttaa kehittämään entistä tehokkaampia ja laadukkaampia sovelluksia tekoälyn avulla.

2 TEKÖÄLYN PERUSTEET JA SOVELTAMINEN OHJELMOINNISSA

2.1 Tekoälyn historia ja perusteet

Tekoäly on tieteen ja teknologian ala, joka kehittää koneita ja ohjelmistoja suorittamaan tehtäviä, joissa yleensä tarvitaan ihmisen älykkyyttä. Tekoälyn historia ulottuu 1950-luvulle, jolloin ensimmäiset ajatukset ja tutkimukset koneiden älykkyydestä saivat alkunsa. Nykypäivänä tekoäly on kehittynyt huomattavasti ja sen sovellusalueet ovat laajentuneet monille eri aloille. (1.)

Tekoäly voidaan jakaa kahteen pääkategoriaan: heikkoon ja vahvaan tekoölyyn. Heikko tekoäly (weak AI, narrow AI) tarkoittaa ohjelmistoja ja algoritmeja, jotka on suunniteltu suorittamaan tiettyjä tehtäviä erittäin hyvin, mutta ne eivät pysty ymmärtämään tai ratkaisemaan ongelmia laajemmin. Vahva tekoäly (strong AI), joka on vielä pitkälti teoreettista, tarkoittaa sellaista tekoölyä, joka kykenee ymmärtämään ja ratkaisemaan laajoja ongelmia ja suorittamaan tehtäviä samalla tavalla kuin ihminen. (2.)

2.2 Tekoälytekniikat ohjelmoinnissa

Tekoälyä voidaan soveltaa ohjelmoinnissa monin eri tavoin. Tässä osiossa käsitellään joitakin yleisimpiä tekoälytekniikoita, joita voidaan hyödyntää ohjelmoinnin ja sovelluskehityksen kontekstissa.

- Koneoppiminen (machine learning): Koneoppiminen on tekoälyn ala, joka keskittyy algoritmien ja tilastollisten mallien kehittämiseen, joiden avulla tietokoneet voivat oppia ja parantaa suorituskyykyään tehtävissä ilman erillistä ohjelmointia. Koneoppimista voidaan käyttää esimerkiksi koodin analysointiin, virheiden tunnistamiseen ja ohjelmistojen testaukseen. (3; 4.)
- Neuroverkot ja syväoppiminen (neural network, deep learning): Neuroverkot ovat koneoppimisen menetelmiä, jotka perustuvat ihmisen aivojen toimintaa jäljitteleviin rakenteisiin. Syväoppiminen on neuroverkkojen laajennus, joka kykenee oppimaan monimutkaisia ja hierarkkisia piirteitä datasta. Näitä tekniikoita voidaan hyödyntää esimerkiksi kuvan- ja puheentunnistuksessa, luonnollisen kielen käsittelyssä ja monimutkaisten mallien luomisessa ohjelmoinnin kontekstissa. (5; 6.)

- Luonnollisen kielen käsittely (Natural Language Processing, NLP): NLP on tekoälyn ala, joka keskittyy ihmisen kielen ymmärtämiseen ja tuottamiseen tietokoneilla. NLP:llä voidaan kehittää työkaluja ja sovelluksia, jotka auttavat ohjelmoijia muun muassa koodin dokumentoinnissa, koodin kommenttien analysoinnissa ja ohjelmistojen käyttöliittymien suunnittelussa. (7; 8.)
- Asiantuntijajärjestelmät (expert system): Asiantuntijajärjestelmät ovat tekoälyohjelmistoja, jotka on suunniteltu ratkaisemaan ongelmia tietyllä asiantuntemuksen alueella. Ne pohjautuvat tiettyjen sääntöjen ja logiikan soveltamiseen ongelmanratkaisussa. Asiantuntijajärjestelmiä voidaan käyttää esimerkiksi ohjelmoinnin apuna tarjoamaan suosituksia koodin optimoinnista, arkkitehtuurista ja parhaista käytännöistä. (9; 10.)
- Hahmontunnistus ja tiedonlouhinta (pattern recognition, data mining): Hahmontunnistus on tekoälyn ala, joka keskittyy automaattiseen tunnistamiseen ja luokitteluun erilaisten mallien ja piirteiden perusteella. Tiedonlouhinta puolestaan tarkoittaa prosessia, jossa pyritään löytämään ja erottamaan hyödyllistä tietoa suurista tietomääristä. Näitä tekniikoita voidaan hyödyntää ohjelmoinnin yhteydessä esimerkiksi koodin analysoinnissa, virheiden havaitsemisessa ja suorituskyvyn parantamisessa. (11; 12.)

2.3 Tekoälyn soveltaminen Android-sovellusten kehittämisessä

Tekoäly voi tarjota monia hyötyjä ja mahdollisuuksia Android-sovellusten kehittämisessä. Seuraavassa on lueteltu joitakin tapoja, joilla tekoälyä voidaan hyödyntää sovelluskehityksen eri vaiheissa.

- Suunnittelu: Tekoälyä voidaan käyttää käyttöliittymien ja käyttökokemusten suunnittelussa. Esimerkiksi luonnollisen kielen käsittelyn avulla voidaan analysoida käyttäjien palautetta ja ehdottaa parannuksia sovelluksen suunnitteluun. (13.)
- Ohjelmointi: Tekoälypohjaiset työkalut voivat auttaa ohjelmoijia kirjoittamaan tehokasta ja virheetöntä koodia nopeammin. Esimerkiksi koodin täydennystä ja refaktorointia helpottavat työkalut voivat perustua tekoälyyn ja koneoppimiseen, mikä parantaa kehitysprosessin tehokkuutta. (14.)

- Testaus: Tekoälyä voidaan käyttää automatisoidun testauksen parantamiseen ja laajentamiseen. Koneoppiminen ja hahmontunnistus voivat auttaa tunnistamaan virheitä ja suorituskykyongelmia koodissa, mikä mahdollistaa niiden nopeamman korjaamisen. (15.)
- Optimointi: Tekoälypohjaisten analytiikkatyökalujen avulla voidaan tunnistaa suorituskykyyn ja energiatehokkuuteen liittyviä ongelmia, jolloin kehittäjät voivat optimoida sovelluksen toimintaa ja resurssien käyttöä. (14.)
- Julkaisu ja markkinointi: Tekoälyä voidaan käyttää sovelluksen julkaisuprosessin tehostamiseen ja markkinointitoimenpiteiden kohdentamiseen. Esimerkiksi luonnollisen kielen käsittelyä voidaan käyttää arvioimaan sovelluksen arvosteluja ja palautteita, jotta kehittäjät voivat reagoida nopeasti käyttäjien tarpeisiin ja toiveisiin. (16.)

Näiden esimerkkien avulla on helppo nähdä, kuinka tekoäly voi tarjota monia hyötyjä ja mahdollisuuksia Android-sovellusten kehityksessä.

3 ANDROID-SOVELLUSTEN KEHITTÄMISEN PERIAATTEET, TYÖKALUT JA PARHAAT KÄYTÄNNÖT

3.1 Android-sovelluskehityksen periaatteet

Android on avoimen lähdekoodin mobiilikäyttöjärjestelmä, jonka on kehittänyt Google. Android-sovelluskehityksen periaatteet keskittyvät sovellusten luomiseen, jotka ovat käyttäjäystävällisiä, toimivia ja joustavia eri laitteille ja näyttökoolle. Seuraavassa on esitetty joitakin tärkeitä periaatteita, jotka tulisi ottaa huomioon Android-sovelluksia kehitettäessä (17):

- **Käyttäjakeskeisyys:** Android-sovellusten tulisi olla suunniteltu käyttäjän tarpeet ja mieltymykset huomioon ottaen. Käyttäjäkokemus on tärkeä tekijä sovelluksen menestyksessä, ja sen tulisi olla helppokäyttöinen, intuitiivinen ja visuaalisesti miellyttävä.
- **Yhteensopivuus:** Android-laitteita on monenlaisia, ja niiden näyttökoot, resoluutiot ja laitteisto-resurssit vaihtelevat. Sovelluksen tulisi olla yhteensopiva erilaisten laitteiden kanssa ja skaalautua joustavasti eri näyttökokojen ja resoluutioiden mukaan.
- **Suorituskyky:** Hyvä sovellus toimii tehokkaasti ja nopeasti, tarjoaa sujuvan käyttökokemuksen ja minimoi laitteen resurssien kulutuksen. Kehittäjien tulisi kiinnittää huomiota sovelluksen suorituskykyyn ja optimoida sen toimintaa niin, että se toimii hyvin erilaisilla laitteilla ja verkkoyhteyksillä.
- **Tietoturva:** Android-sovellusten kehittäjien on huolehdittava tietoturvasta ja suojattava käyttäjien tietoja ja yksityisyyttä. Tämä tarkoittaa esimerkiksi tiedonsiirron salaamista, käyttäjätietojen säilyttämistä turvallisesti ja sovelluksen suojaamista haittaohjelmilta ja hyökkäyksiltä.

3.2 Android-sovelluskehityksen työkalut

Android-sovellusten kehittämiseen on olemassa monia erilaisia työkaluja ja resursseja. Tässä osiossa käsitellään joitakin yleisimpiä työkaluja, jotka helpottavat sovellusten suunnittelua, koodausta, testausta ja julkaisua:

- **Android Studio:** Android Studio on Googlen virallinen integroitu kehitysympäristö (IDE) Android-sovellusten kehittämiseen. Ohjelmassa on kattava valikoima työkaluja ja ominaisuuksia, kuten koodin muokkausta, virheenkorjausta, suorituskyvyn profilointia ja laaja-alaista tukea Android-laitteille. Android Studio sisältää myös visuaalisen käyttöliittymän suunnittelun työkaluja, joiden avulla voi luoda ja muokata näkymiä helposti (18.)
- **Gradle:** Gradle on avoimen lähdekoodin automaatiojärjestelmä, joka on integroitu Android Studioon. Se helpottaa sovellusprojektien rakentamista ja hallintaa, kuten esimerkiksi riippuvuuksien hallintaa, koodin kääntämistä ja pakkaamista sekä testien suorittamista. (19.)
- **Firebase:** Firebase on Googlen tarjoama pilvipalvelualusta, joka tarjoaa monia työkaluja ja palveluja Android-sovellusten kehittämiseen. Näitä ovat esimerkiksi reaaliaikainen tietokanta, analytiikka, push-ilmoitukset ja käyttäjäautentikointi. (20.)
- **GitHub ja Git:** GitHub on suosittu koodinhallintapalvelu, joka perustuu Git-versionhallintajärjestelmään. Se tarjoaa helpon tavan tallentaa, jakaa ja ylläpitää sovellusprojektien koodia sekä yhteistyö- ja integrointiominaisuuksia muiden kehitystyökalujen kanssa (21; 22.)
- **Espresso ja UI Automator:** Espresso ja UI Automator ovat Android-sovellusten testaustyökaluja, jotka keskittyvät käyttöliittymän testaukseen ja automatisointiin. Ne auttavat varmistamaan, että sovellus toimii odotetusti eri laitteilla ja käyttöolosuhteissa. (23; 24.)

3.3 Parhaat käytännöt Android-sovelluskehityksessä

Hyvien käytäntöjen noudattaminen Android-sovellusten kehityksessä voi auttaa luomaan laadukkaampia ja menestyksekkäämpiä sovelluksia. Tässä osiossa esitellään joitakin suosituksia, jotka kannattaa ottaa huomioon sovelluskehityksen aikana:

- **Suunnittele ennen koodausta:** Sovelluksen suunnitteluvaiheessa on tärkeää määritellä sen tavoitteet, käyttötapaukset ja käyttäjäkokemus. Tämä auttaa varmistamaan, että sovellus on käyttäjäystävällinen ja vastaa käyttäjien tarpeisiin. (25.)

- Noudatetaan Androidin suunnitteluperiaatteita: Androidin suunnitteluperiaatteiden noudattaminen auttaa luomaan yhtenäisen ja helppokäyttöisen käyttöliittymän sovellukselle. Google tarjoaa Androidin suunnitteluohjeet (Material Design), jotka kannattaa ottaa huomioon sovelluksen ulkoasun ja toiminnan suunnittelussa. (26.)
- Modularisoi koodi: Koodin modularisointi tarkoittaa sen jakamista pienempiin, itsenäisiin osiin tai moduuleihin. Tämä tekee koodista helpommin ymmärrettävää, ylläpidettävää ja testattavaa. Hyödynnä objektiivista ohjelmointia, jaota toiminnallisuus luokkiin ja käytä rajapintoja moduulien väliseen kommunikointiin. (27.)
- Käytä versionhallintaa: Versionhallinnan käyttö on tärkeää sovellusprojektin koodin hallinnassa ja ylläpidossa. Se mahdollistaa muutosten seuraamisen, vanhempien versioiden palauttamisen ja yhteistyön muiden kehittäjien kanssa. Git on yleisesti käytetty versionhallintajärjestelmä, ja GitHub tai GitLab tarjoavat pilvipohjaisia palveluita koodin tallentamiseen ja jakamiseen. (28.)
- Testaa sovellusta kattavasti: Sovelluksen testaaminen on olennainen osa kehitysprosessia, ja sen avulla voidaan varmistaa, että sovellus toimii oikein ja täyttää käyttäjien odotukset. Kattavaan testaamiseen käytetään erilaisia testaustekniikoita, kuten yksikkötestausta, integraatio-testausta ja käyttöliittymätestausta, jotta voit tunnistaa ja korjata mahdolliset virheet ja ongelmat. (29.)
- Pidä huolta tietoturvasta ja yksityisyydestä: Kehitä sovelluksia turvallisuus- ja yksityisyysnäkökohdat huomioon ottaen. Tiedon siirtoon käytetään salattuja yhteyksiä, käyttäjien henkilötiedot säilytetään turvallisesti ja pyydetään vain välttämättömät käyttöoikeudet sovelluksen toiminnan kannalta. (30.)

Nämä ovat vain joitakin esimerkkejä parhaista käytännöistä, jotka auttavat luomaan laadukkaita ja menestyksekkäitä Android-sovelluksia. Tekoälyn ja koneoppimisen hyödyntäminen kehitysprosessin eri vaiheissa voivat edelleen parantaa sovellusten suunnittelua, suorituskykyä ja käyttäjäkoke-
musta.

4 TEKÖÄLYN HYÖDYNTÄMINEN ANDROID-SOVELLUKSISSA

Tekoäly ja koneoppiminen ovat nopeasti kehittyviä teknologioita, jotka tarjoavat uusia mahdollisuuksia ja ratkaisuja monille eri aloille, mukaan lukien mobiilisovellusten kehitys. Tässä luvussa tarkastelemme, kuinka tekoälyä voidaan hyödyntää osana Android-sovellusta ja esittelemme joitakin konkreettisia esimerkkejä tekoälypohjaisten sovellusten luomisesta.

4.1 Koneoppimisen integroiminen Android-sovelluksiin

Google on julkaissut useita työkaluja ja palveluita, jotka helpottavat tekoälyn ja koneoppimisen integroimista Android-sovelluksiin. Yksi tällainen työkalu on ML Kit (31), joka tarjoaa valmiita koneoppimismalleja ja -ratkaisuja yleisiin tehtäviin, kuten kuvantunnistus, tekstintunnistus ja kasvojen tunnistus. ML Kitin avulla kehittäjät voivat helposti lisätä tekoälyominaisuuksia sovelluksiinsa ilman syvällistä koneoppimisen asiantuntemusta.

TensorFlow Lite (32) on kevyt ja tehokas työkalu, joka on suunniteltu erityisesti mobiililaitteille ja sulautetuille järjestelmille. Se mahdollistaa kehittäjien luoda ja kouluttaa omia koneoppimismallejaan ja integroida ne Android-sovelluksiin. TensorFlow Liten avulla kehittäjät voivat rakentaa monimutkaisempia ja räätälöityjä tekoälyratkaisuja, jotka vastaavat heidän sovellustensa erityistarpeita.

4.2 Esimerkkejä tekoälypohjaisista Android-sovelluksista

Tekoälypohjaiset sovellukset voivat parantaa käyttäjäkokemusta ja tarjota arvokkaita uusia toimintoja monilla eri alueilla. Tässä on muutamia esimerkkejä tekoälypohjaisista Android-sovelluksista:

- Kuvantunnistus ja kuvankäsittely: Sovellukset, kuten Google Lens (33), hyödyntävät tekoälyä tunnistamaan kuvien kohteita, tekstejä ja muita yksityiskohtia. Tämä voi auttaa käyttäjiä tunnistamaan tuotteita, löytämään tietoja nähtävyyksistä tai jopa kääntämään vieraskielisiä tekstejä reaaliajassa.
- Ääniohjaus ja puheentunnistus: Tekoälyä voidaan käyttää ääniohjauksen ja puheentunnistuksen parantamiseen sovelluksissa, kuten Google Assistant (34), joka kykenee ymmärtämään

käyttäjän puhetta ja suorittamaan pyydettyjä toimintoja. Tämä voi helpottaa sovellusten käyttöä ja parantaa käyttäjäkokemusta erityisesti silloin, kun käsien käyttö ei ole mahdollista tai kätevää.

- Suosittelevärjestelmät: Sovellukset, kuten Spotify (35) ja Netflix (36), hyödyntävät tekoälyä ja koneoppimista analysoimaan käyttäjien tottumuksia ja mieltymyksiä ja tarjoavat heille henkilökohtaisia suosituksia musiikista, elokuvista ja sarjoista. Tämä parantaa käyttäjäkokemusta ja auttaa heitä löytämään uutta sisältöä, josta he todennäköisesti pitävät.
- Tekstin syöttö ja ennustaminen: Tekoälyä voidaan käyttää myös parantamaan tekstinsyöttöä ja ennustamaan seuraavaa sanaa, kuten esimerkiksi SwiftKey-näppäimistösovelluksessa (37). Tämä voi nopeuttaa kirjoittamista ja vähentää virheiden määrää, mikä tekee tekstiviestien ja sähköpostien kirjoittamisesta helpompaa ja tehokkaampaa.
- Laajennettu todellisuus (AR): AR-sovellukset, kuten Google Maps Live View (38), hyödyntävät tekoälyä ja kuvantunnistusta käyttäjän navigointikokemuksen parantamiseen yhdistämällä kameran näkymän karttatietoihin ja tarjoamalla reaaliaikaisia ohjeita.

4.3 Tekoälyn haasteet ja eettiset kysymykset

Vaikka tekoäly tarjoaa monia mahdollisuuksia Android-sovellusten kehittämiseen, se tuo mukanaan myös haasteita ja eettisiä kysymyksiä. Yksi tärkeä haaste on tietosuoja ja yksityisyys, sillä tekoälysovellukset saattavat käsitellä ja tallentaa käyttäjien henkilökohtaisia tietoja. Kehittäjiä on varmistettava, että tällaiset tiedot suojataan asianmukaisesti ja että käyttäjät antavat suostumuksensa niiden käsittelyyn.

Toinen huolenaihe on tekoälyn käyttämien koneoppimismallien puolueettomuus ja syrjimättömyys. Joissakin tapauksissa mallit saattavat olla vinoutuneita tai syrjiviä puutteellisten tai epätasapainoisten koulutusaineistojen takia. Kehittäjiä on kiinnitettävä huomiota mallien koulutukseen ja varmistettava, että ne eivät tuota epäoikeudenmukaisia tai syrjiviä tuloksia (39).

Tekoälyn käyttöön liittyy myös vastuukysymyksiä. Esimerkiksi jos tekoälypohjainen sovellus antaa virheellistä tai haitallista tietoa, kuka on vastuussa tästä? Kehittäjiä on pohdittava näitä kysymyksiä ja opastettava käyttäjiä tekoälyn rajoituksista ja mahdollisista riskeistä.

Kehittäjän on tärkeää ottaa huomioon käyttäjien kokemus ja heidän tarpeensa. Tekoälyn integroiminen sovelluksiin voi parantaa käyttäjäkokemusta, mutta samalla se voi myös tehdä sovelluksista monimutkaisempia ja vaikeampia käyttää. Kehittäjiä on löydettävä tasapaino tekoälyn hyötyjen ja käytettävyyden välillä, jotta sovellukset palvelevat käyttäjiä parhaalla mahdollisella tavalla.

Tekoälyn käyttö Android-sovellusten kehityksessä tarjoaa monia mahdollisuuksia ja haasteita. Kehittäjiä on oltava tietoisia näistä haasteista ja eettisistä kysymyksistä, jotta he voivat luoda turvallisia, käyttäjäystävällisiä ja vastuullisia tekoälypohjaisia sovelluksia. Tämän opinnäytetyön tavoitteena on tutkia näitä kysymyksiä ja tarjota konkreettisia esimerkkejä ja suosituksia tekoälyn hyödyntämiseen Android-sovellusten kehityksessä.

5 ANDROID-SOVELLUKSEN KEHITTÄMINEN KÄYTÄNNÖSSÄ

Tässä luvussa kerrotaan käytännön kokemuksista tekoälyn hyödyntämisestä Android-sovelluksen kehittämisessä. Työn tuloksena syntyi Oulu-aiheinen sovellus, joka hyödyntää Oulunliikenne.fi-palvelun avointa dataa (40) ja Google Maps -karttapalvelua (41). Pyrkimyksenä oli hyödyntää tekoälyä sovelluksen kehittämisen kaikissa vaiheissa.

5.1 Sovelluksen ideointi

Ideoinnissa tekoälyn käyttö jäi vähäisemmäksi ja sisälsi ainoastaan sovellukseen lisäämättä jääneen tekoälymatkaoppaan, joka olisi käyttäjän kehoitteen perusteella ehdottanut sopivia vierailukohteita Oulusta. Sovelluksen perusideana oli käyttää Oulunliikenteen tarjoamia GraphQL-rajapintoja (42) ja näyttää niiden aineistoja kartalla. Avoimen datan aineistosta valitsin kartalla näytettäväksi kolme: Oulun alueen liikennekameroiden kuvat, Oulun pysäköintitalojen ajantasaiset tilatiedot ja liikenteen automaattisten mittauspisteiden (LAM-pisteiden) tuottama data. Valitsin nämä kolme sen takia, että ne eroavat toisistaan siinä, miten vastaanotettua dataa sovelluksessa käytetään. Kuvassa 1 nähdään GraphQL-kyselyn palauttamaa JSON-tyyppistä dataa, josta luetaan koordinaatit ja ne sijoitetaan Google Maps -kartalle.

```

"cameras": [
  {
    "cameraId": "C12501",
    "name": "vt20_Oulu_Arkala",
    "lat": 65.189479,
    "lon": 26.1921,
    "presets": [
      {
        "presetId": "C1250101",
        "presentationName": "Kuusamoon",
        "imageUrl": "https://weathercam.digitraffic.fi/C1250101.jpg",
        "measuredTime": "2023-10-26T15:47:51Z"
      },
      {
        "presetId": "C1250102",
        "presentationName": "Ouluun",
        "imageUrl": "https://weathercam.digitraffic.fi/C1250102.jpg",
        "measuredTime": "2023-10-26T15:48:25Z"
      },
      {
        "presetId": "C1250107",
        "presentationName": "Tienpinta2",
        "imageUrl": "https://weathercam.digitraffic.fi/C1250107.jpg",
        "measuredTime": "2023-10-26T15:48:13Z"
      },
      {
        "presetId": "C1250109",
        "presentationName": "Tienpinta",
        "imageUrl": "https://weathercam.digitraffic.fi/C1250109.jpg",
        "measuredTime": "2023-10-26T15:47:45Z"
      }
    ]
  }
],

```

KUVA 1. JSON-dataa yhdestä kamerasijainnista

Karttamerkin valitsemalla käyttäjä saa haluamansa sijainnin tiedot näkyviin, Liikennekameran valinta listaa sijainnin kameralta ja niiden kuvat. Pysäköintitalojen sijainnin lisäksi kartalla näkyvät vapaat paikat ja kohteen valitsemalla käyttäjä saa näkyviin pysäköintitalon nimen ja paikkatilanteen. LAM-pisteen valinta listaa kaikki sijainnin mittaustiedot.

5.2 Sovelluksen suunnittelu

Suunnittelussa tekoälyä käytettiin reilusti enemmän kuin ideointivaiheessa. Ideoinnin pohjalta suunniteltiin Android-sovellukseen kolme eri aktiviteettia, yksi jokaiselle näytettävälle datalle sekä etusivu, missä käyttäjä valitsee, mitä dataa haluaa nähdä. Suunnittelu tapahtui osittain samaan aikaan ohjelmoinnin kanssa, esimerkiksi sopivia kirjastoja valittaessa. Tekoäly antoi välillä turhan monimutkaisia ratkaisuja, jotka käyttivät useita eri kirjastoja. Esimerkiksi GraphQL-kyselyjen tekoon ensimmäinen ehdotus oli käyttää Apollo Client -tilanhallintakirjastoa (43), mutta koska sovelluksen tekemät kyselyt eivät ole kovin monimutkaisia, päädyttiin käyttämään yksinkertaista HTTP-pyyntöä, OkHttp3-kirjastoa (44) hyödyntäen.

5.3 Sovelluksen ohjelmointi ja käytetyt tekoälytyökalut

Tässä luvussa kerrotaan käytännön kokemuksista eri tekoälytyökalujen käyttämisestä ohjelmoinnissa. Ohjelmointi oli projektin vaihe, jossa tekoälytyökaluja käytettiin selkeästi eniten. Tekoäly kykenee tuottamaan suuriakin koodikokonaisuuksia kerralla, mutta parempaan lopputulokseen pääsee yleensä jakamalla kysymykset pienempiin koodin osiin sekä virheiden korjaamiseen.

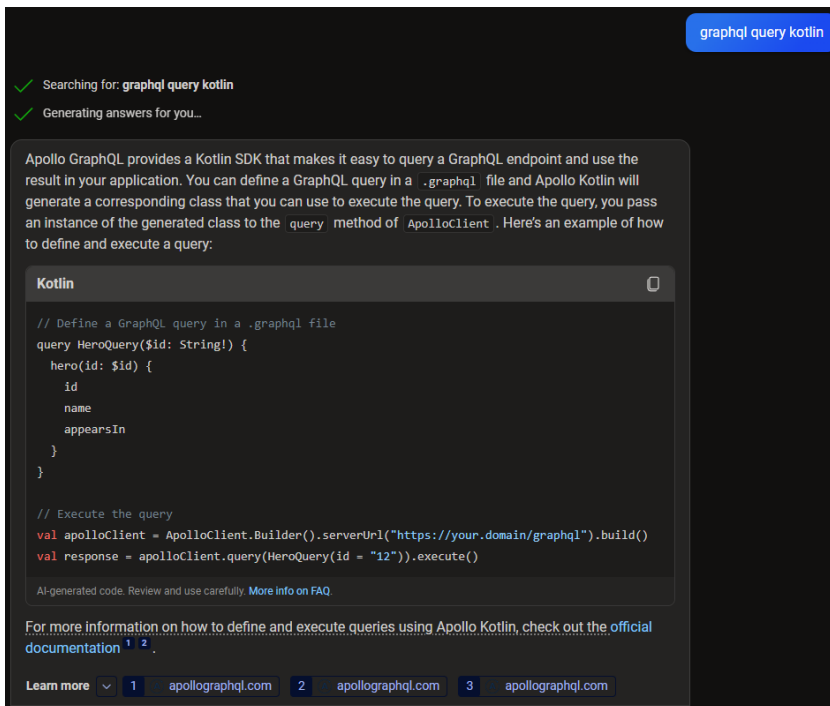
Ohjelmointikielien ja kirjastojen ovat muuttuneet vuosien varrella, ja varsinkin pidemmissä ja monimutkaisemmissa tekoälyn tuottamissa koodeissa on usein käytöstä poistuneita termejä, ominaisuuksia ja syntakseja. Yhdellä kysymyksellä tekoälyltä saatu koodi saattaa sisältää niin paljon ongelmia, että sen korjaaminen toimivaksi vaatii enemmän aikaa kuin koodissa tarvittavien osien selvittäminen ja kokonaisuuden rakentaminen itse.

5.3.1 GPT-4

GPT-4 (45) on OpenAI:n kehittämä syväoppimiseen perustuva kielimalli, joka on koulutettu noin 10 biljoonan sanan suuruisella datamäärällä (46). Multimodaalisena mallina se pystyy ottamaan syötteenä vastaan sekä tekstiä että kuvia ja tuottamaan ihmismäistä tekstiä ja päättelyä. GPT-4 on monilla mittareilla ihmisen tasolla ja läpäisee esimerkiksi simuloidun lakitutkinnon paremmilla pisteillä kuin 90 % testin suorittajista (45). Kyvykkyydestään huolimatta sillä on myös rajoituksia, joista tärkeimpänä epäluotettavuus johtuen esimerkiksi sen taipumuksesta hallusinoita faktoja (47).

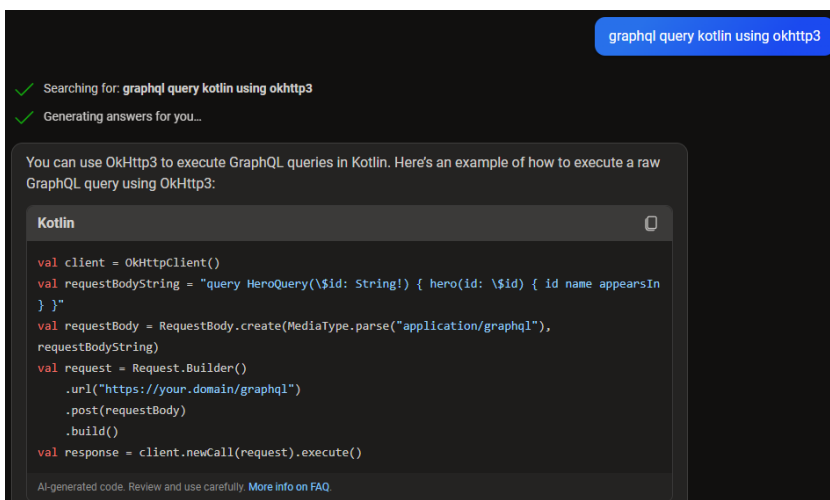
5.3.2 ChatGPT ja Bing Chat

OpenAI:n GPT-4:ää käyttävät Bing-hakukoneen (48) Bing Chat (49) ja OpenAI:n oma ChatGPT (50) ovat hyviä työkaluja ongelmien ratkaisuun ja esimerkkikoodien luontiin. ChatGPT ja Bing Chat eroavat toisistaan lähinnä siinä, että Bing Chatin keskustelut on rajattu 4000 merkkiin ja 30 viestiin. Maksullisessa ChatGPT Plus -palvelussa (51) rajoitukset ovat reilusti suuremmat ja koskevat kokonaista keskustelua eivätkä yksittäisiä kehoitteita. Seuraavana on esimerkki GraphQL-kyselyn suorittavan koodin toteuttamisesta Bing Chatin avulla. Bing Chatin tekoäly tuottaa koodia hyvin yksinkertaisillakin kysymyksillä. Esimerkiksi syötteellä "graphql query kotlin" tekoäly tuottaa kuvassa 2 näkyvän vastauksen, joka sisältää Kotlin-kielisen koodin GraphQL-kyselyn suorittamiseen ApolloClient kirjastoa käyttäen.



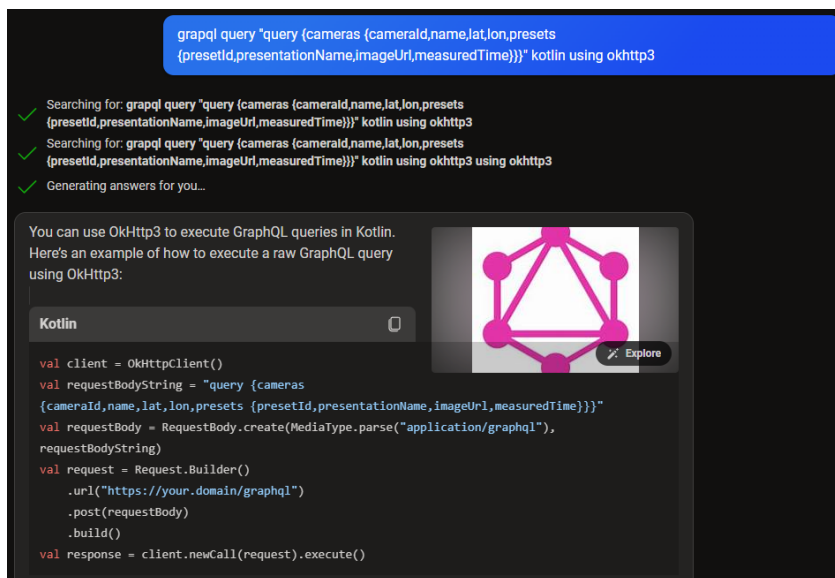
KUVA 2. Bing Chatin vastaus ensimmäiseen kyselyyn ”graphql query kotlin”

Syy sille, että tekoälyn vastauksessa kirjastoksi on valikoitunut juuri ApolloClient on yhdistelmä sattumaa ja sen suosiota materiaalissa, jolla GPT-4:n kielimalli on koulutettu. Koska tarkoituksena on suorittaa kysely käyttämättä ApolloClient-kirjastoa pelkällä http-pyyntöllä, syötteeseen lisätään kirjasto, jota halutaan käyttää. Uusi kuvan 3 mukainen kysely ”graphql query kotlin using okhttp3” tuottaa oikeaa kirjastoa käyttävän Kotlin-koodin.



KUVA 3. Bing Chatin vastaus käytettävällä kirjastolla tarkennettuun kyselyyn

Jos kysely suoritetaan okhttp3 sijasta pelkällä http-termillä, antaa se edelliseen kyselyyn verrattuna paljon useammin muita kirjastoja kuin OkHttp3. Projektissa käytetyt kirjastot, OkHttp3:a lukuun ottamatta, valittiin käyttöön ChatGPT:n ehdotuksien perusteella. Tavoitteena oli saada tekoälyltä sovellukseen sopiva ja toimiva koodi, joten tarkennetaan syötettä vielä kerran kuvassa 4 esitetyllä tavalla lisäämällä siihen GraphQL-kyselyn "grapql query "query {cameras {cameraId,name,lat,lon,presets {pre-setId,presentationName,imageUrl,measuredTime}}}" kotlin using okhttp3".



KUVA 4. Bing Chatin vastaus graphql-kyselyn sisällöllä tarkennettuun kyselyyn

Huomioitavaa on, että kysely toimii oikein, vaikka siinä on kirjoitusvirhe: "grapql" pitäisi olla "graphql". Tämä johtuu siitä, että tekoäly osaa korjata pienet virheet kyselyssä. Koska Oulunliikenteen GraphQL-palvelu palauttaa JSON-tyyppistä dataa, tarvitsee syötettä vielä tarkentaa kuten kuvassa 5.

grapql query for JSON "query {cameras {cameraId,name,lat,lon, presets {presetId,presentationName,imageUrl,measuredTime}}}" kotlin using okhttp3

- ✓ Searching for: grapql query for JSON "query {cameras {cameraId,name,lat,lon, presets {presetId,presentationName,imageUrl,measuredTime}}}" kotlin using okhttp3
- ✓ Searching for: grapql query for JSON "query {cameras {cameraId,name,lat,lon, presets {presetId,presentationName,imageUrl,measuredTime}}}" kotlin using okhttp3 okhttp3
- ✓ Generating answers for you...

Here is an example of a **Kotlin** code snippet that uses **OkHttp3** library to execute a GraphQL query and retrieve the JSON response:

```

Kotlin
import okhttp3.MediaType.Companion.toMediaType
import okhttp3.OkHttpClient
import okhttp3.Request
import okhttp3.RequestBody.Companion.toRequestBody

fun main() {
    val query = """
        query {
            cameras {
                cameraId
                name
                lat
                lon
                presets {
                    presetId
                    presentationName
                    imageUrl
                    measuredTime
                }
            }
        }
    """.trimIndent()

    val mediaType = "application/json; charset=utf-8".toMediaType()
    val requestBody = query.toRequestBody(mediaType)

    val request = Request.Builder()
        .url("https://your-graphql-api.com")
        .post(requestBody)
        .build()

    val client = OkHttpClient()
    val response = client.newCall(request).execute()

    if (response.isSuccessful) {
        val json = response.body?.string()
        println(json)
    } else {
        println("Error: ${response.code} ${response.message}")
    }
}

```

AI-generated code. Review and use carefully. [More info on FAQ](#)

This code snippet sends a GraphQL query to the specified URL and retrieves the JSON response using OkHttp3 library. You can replace the `query` variable with your own GraphQL query and modify the URL to match your API endpoint.

I hope this helps! Let me know if you have any other questions.

Learn more [1](#) [2](#) [3](#)

KUVA 5. Bing Chatin vastaus palautettavan datan tyypillä tarkennettuun kyselyyn

Tekoälyn tuottama koodi toimii vasta, kun kyselyn (query) rakennetta on hieman korjattu ja paikka-merkkinä oleva osoite "https://your-graphql-api.com", on korvattu Oulunliikenteen avoimen datan GraphQL URL-osoitteella "https://api.ouunliikenne.fi/proxy/graphql/".

ChatGPT:n luoma koodi ei välttämättä ole yhteensopiva kehitteillä olevan ohjelman kanssa, eikä sen viimeistely ChatGPT:lle annettua kehoitetta muokkaamalla ole helppoa tai edes mahdollista,

varsinkaan, jos kehoitteeseen ei haluta lisätä muuta projektin koodia. Koodin viimeistely tekoälytyökaluja käyttäen voidaan tehdä Android Studiossa tai vastaavassa koodieditorissa käyttäen apuna esimerkiksi GitHub Copilota.

5.3.3 GitHub Copilot

Android Studioon (52), Visua Studio Codeen (53), Visual Studioon ja muutamisiin muihin koodieditoreihin löytyvä GitHub Copilot (54) -laajennus helpottaa ja nopeuttaa koodin kirjoittamista. OpenAI:n ja GitHubin yhteistyönä GPT-mallista kehitetty ja GitHubin julkisilla koodivarastoilla koulutettu suuri hermoverkko GitHub Copilot on tekoälyavusteinen pariohjelmoija, joka tarjoaa automaattisia ehdotuksia koodista ja kommentteista, jotka käyttäjä voi hyväksyä tai jättää huomiotta koodia kirjoittaessaan.

GitHub Copilot on erityisen hyödyllinen silloin, kun koodi sisältää useita samanlaisia toistuvia rakenteita. GitHub Copilot käyttää kuvassa 6 näkyvää projektin olemassa olevaa koodia kelikameroiden kuvien dataluokasta, jota käytetään GraphQL-kyselyn palauttaman datan säilytykseen ja ennustaa asiayhteyden perusteella, mitä vastaavan kuvassa 7 olevan parkkitalojen paikkatilanteen dataluokan tulee siihen liittyvän GraphQL-kyselyn perusteella sisältää.

```
val query =
    "query {cameras {cameraId,name,lat,lon,presets {presetId,presentationName,imageUrl,measuredTime}}}"
val client = OkHttpClient()
val json = JSONObject()
json.put(name="query", query)
val mediaType = "application/json; charset=utf-8".toMediaTypeOrNull()
val requestBody = json.toString().toRequestBody(mediaType)
val request = Request.Builder()
    .url("https://api.oulunliikenne.fi/proxy/graphql/")
    .post(requestBody)
    .build()

data class Camera(
    val name: String,
    val lat: Double,
    val lon: Double,
    val imageUrls: List<String>,
    val presentationNames: List<String>,
    val measuredTime: String
)
```

KUVA 6. Kameroiden tietoja hakeva GraphQL-kysely

```

val query =
    "query {carParks {carParkId,name,lat,lon,maxCapacity,spacesAvailable}}"
val client = OkHttpClient()
val json = JSONObject()
json.put(name: "query", query)
val mediaType = "application/json; charset=utf-8".toMediaTypeOrNull()
val requestBody = json.toString().toRequestBody(mediaType)
val request = Request.Builder()
    .url("https://api.oulunliikenne.fi/proxy/graphql/")
    .post(requestBody)
    .build()

data class CarPark(
    val carParkId: String,
    val name: String,
    val lat: Double,
    val lon: Double,
    val maxCapacity: Int,
    val spacesAvailable: Int
)

```

KUVA 7. GitHub Copilotin tuottama dataluokka

Toinen esimerkki GitHub Copilotin toistuvien rakenteiden ennustamisesta on kuvassa 8 näkyvä tilanne, jossa useita eri muuttujia käsitellään samalla tavoin.

```

smParams.weight = 1.2f
sm.layoutParams = smParams
mvParams.weight = 0.8f
mv.layoutParams = mvParams

```

KUVA 8. GitHub Copilotin tuottama koodirivi.

GitHub Copilot kykenee myös ChatGPT:n tapaan tuottamaan koodiehdotuksia annetun syötteen perusteella, kuten esimerkki kuvassa 9, jossa käyttäjä on kirjoittanut kuvauksen silmukasta, joka tulostaa numerot 1–100 ja merkitsee alkuluvut tähdellä.

```
//a loop that prints out numbers from 1 to 100 and marks prime numbers with a star
/*for (i in 1..100) {
  var isPrime = true
  for (j in 2 until i) {
    if (i % j == 0) {
      isPrime = false
      break
    }
  }
  if (isPrime) {
    println("$i*")
  } else {
    println(i)
  }
}*/
```

KUVA 9. GitHub Copilotin kehoitteen perusteella tuottama koodi

5.3.4 Ohjelmointi käytännössä

Ennen tämän projektin aloittamista oli Oulunliikenteen avoimeen dataan tutustuttu jo aikaisemmin ohjelmoimalla GraphQL-kyselyitä tekevä, kelikameroiden kuvia lataava Python-skripti, joten täysin tyhjästä ei aloitettu vaan jonkinlainen käsitys sovelluksen koodin sisällöstä oli jo ennakkoon. Tekoälyä käytettiin apuna koko ohjelmointiprosessin läpi koodin tuottamiseen ja ongelmien ratkaisemiseen. Uusien ominaisuuksien ohjelmointi aloitettiin kysymällä tekoälyltä sopivaa esimerkki koodia, ensisijaisesti ChatGPT:ltä mutta myös GitHub Copilot oli käytössä.

Ohjelmointi aloitettiin kuvassa 10 näkyvän pääsivun (main activity) tekemisellä, johon laitettiin taustalle sivuttain vieritettävä Oulun kaupungin englanninkielisen matkailusivuston (<https://visi-toulu.fi/en/>) taustakuva. "Welcome to Oulu"- teksti avaa matkailusivuston verkkoselaimessa ja yläkulmassa oleva tekstimuotoinen painike pysäyttää automaattisen vierityksen. Käyttäjä voi halutesaan liikuttaa taustakuvaa manuaalisesti.



KUVA 10. Sovelluksen pääsivu

Vierittämisen kuvassa 11 näkyvä koodi rakennettiin inkrementaalisesti tekoälyn kehoitetta tarkentamalla aina, kun edellinen oli varmistettu toimivaksi.

```

val scrolling = findViewById<HorizontalScrollView>(R.id.scrollingImage)
var scrollSpeed = 1
var scrollDirection = scrollSpeed
val scrollRunnable = object : Runnable {
    override fun run() {
        scrolling.smoothScrollBy(scrollDirection * scrollSpeed, 0)
        if (scrollSpeed != 0 && (scrolling.scrollX == 0 || scrolling.scrollX == scrolling.getChildAt(
            0
        ).width - scrolling.width)
        ) {
            scrollDirection = -scrollDirection
        }

        scrolling.postDelayed(this, 50)
    }
}

scrolling.postDelayed(scrollRunnable, 50)

val stopS = findViewById<TextView>(R.id.stopScrolling)
stopS.text = getString(R.string.stopScrolling)
stopS.setOnClickListener {
    if (scrollSpeed == 1) {
        scrollSpeed = 0
        stopS.text = getString(R.string.continueScrolling)
    } else {
        scrollSpeed = 1
        stopS.text = getString(R.string.stopScrolling)
    }
}
}

```

KUVA 11. Sovelluksen pääsivun taustakuvan vierityksen koodi

Ensin taustasta tehtiin käyttäjän liikuteltava, lisättiin automaattinen vieritys, vierityksen suunnanvaihto ja lopuksi mahdollisuus pysäyttää ja jatkaa automaattista vieritystä. Tämän lisäksi etusivulle ei tarvittu muita UI-elementtejä kuin dataa näyttäviä aktiviteettejä aukaisevat painikkeet.

Ensimmäinen kappale Oulusta kertovasta tekstistä on vakio, mutta toinen kappale arvotaan lyhyehköstä listasta erilaisia tekstejä. Trivia-tekstit lisättiin merkkijono resursseihin "values/strings/strings.xml", mistä pääsivun koodi hakee "trivia1", "trivia2" jne. nimetyt merkkijonot listaksi ja arpoo niistä yhden. Tekstien hakemiseen käytetty koodi kävi läpi muutamia eri versioita. Ensimmäinen versio kävi läpi vain koodissa annetun määrän merkkijonoja eli uuden trivian lisääminen vaatisi myös koodin muokkaamista, joten tekoälyn kehotetta tarkennettiin. Lopullinen koodi käy läpi numeroituja trivia-merkkijonoja siihen asti, kun järjestyksessä seuraavaa kokonaislukua ei löydy. Etusivun ja arvottavan tekstin voi päivittää kuvan 12 koodilla ruudun yläreunasta alaspäin vetämällä.

```

val triviaList = mutableListOf<String>()
var i = 1
while (true) {
    val resId = resources.getIdentifier("trivia$i", defType: "string", packageName)
    val trivia = try {
        getString(resId)
    } catch (e: Resources.NotFoundException) {
        break
    }
    triviaList.add(trivia)
    i++
}

val ti = findViewById<TextView>(R.id.textInfo)

fun updateTrivia() {
    val randomTrivia = triviaList.random()
    val triviaText = getString(R.string.info) + "\n\n" + randomTrivia
    ti.text = triviaText
}

updateTrivia()

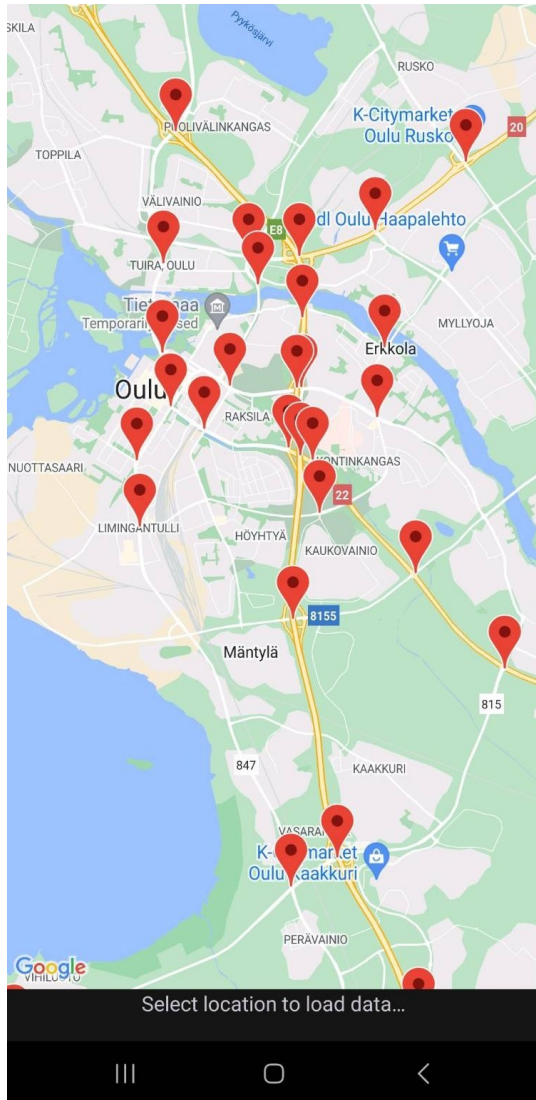
val pullToRefresh = findViewById<SwipeRefreshLayout>(R.id.pullToRefresh)
pullToRefresh.setOnRefreshListener {
    updateTrivia()
    pullToRefresh.isRefreshing = false
}

```

KUVA 12. Pääsivun trivian arvonta ja manuaalisen päivittämisen koodi

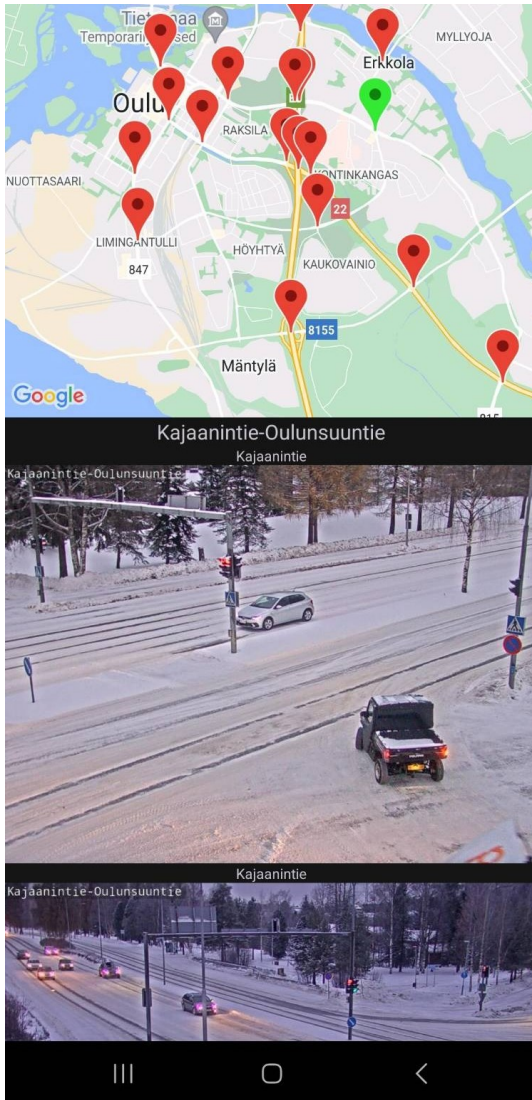
Kartan ja tietojen näyttämisen toiminnot olivat sisällöltään kohtuullisen samanlaisia, joten oli järkevintä tehdä ensin yksi mahdollisimman valmiiksi ja käyttää sitä pohjana muille toiminnoille. Ensimmäiseksi koodattavaksi aktiviteetiksi valikoitui kelikameroiden kuvien katselu, sillä se oli kokonaisuutena monimutkaisin ja sisälsi käytännössä kaiken, mitä muissa toiminnoissa tarvitaan. Tässäkin vaiheessa kaikki koodi tuotettiin tekoälyllä, mutta koodin kompleksisuuden vuoksi se vaati kohtalaisen määrän manuaalisia korjauksia lukuisten ChatGPT-kyselyiden lisäksi.

Kamerakuvien katselutoiminnon valinta avaa kuvassa 13 näkyvän lähes koko ruudun kokoisen Google Maps -kartan, mihin sijoitetaan karttamerkit Oulunliikenteen avoimen datan palvelusta saatujen kameroiden koordinaatit.



KUVA 13. Kelikameroiden sijainteja näyttävä kartta

Karttamerkin valinta pienentää karttanäkymän osuuden puhelimen ruudusta, ja suurentaa alareunassa olevan vieritysnäkymän (scrollview), mihin ladataan sijainnin sisältämien kameroiden nimet ja kuvat, kuvassa 14 näkyvällä tavalla.



KUVA 14. Kerasijainnin valinnan jälkeinen näkymä

Kamera-, parkkitalo- ja mittaustietojentointojen avautuessa kartan osuus kuvaruudusta on 95 prosenttia ja vieritysnäkymä loput 5 prosenttia. Näkymien osuudet on esitetty kuvassa 15 näkyvässä koodissa kahden eri näkymän painoarvoina ja kartamerkin valinnan jälkeen ne ovat kuvassa 8 näkyvän koodin mukaisesti kartalle 40 % ja vieritysnäkymälle 60 %.

```

val sm = findViewById<ScrollView>(R.id.scrollMap)
val smParams = sm.layoutParams as LinearLayout.LayoutParams
smParams.weight = 0.1f
sm.layoutParams = smParams

val mv = findViewById<MapView>(R.id.mapView)
val mvParams = mv.layoutParams as LinearLayout.LayoutParams
mvParams.weight = 1.9f
mv.layoutParams = mvParams

```

KUVA 15. Kartan ja vieritysnäkymän osuudet ruudusta määrittävä koodi

Karttamerkin valinta vaihtaa valitun merkin värin oranssista vihreäksi, että käyttäjä näkee myös kartalla valitsemansa sijainnin. Karttamerkkien värien vaihtamiseen ja niiden näyttämiseen kartalla käytetään kuvassa 16 olevassa koodissa Googlen omia kirjastoja ja kameranäkymän kuvien näyttämiseen puolestaan Glide Image Loader -kirjastoa.

```
runOnUiThread {
    val mapView = findViewById<MapView>(R.id.mapView)
    mapView.getMapAsync { googleMap ->
        for (camera in cameras) {
            val position = LatLng(camera.lat, camera.lon)
            googleMap.addMarker(
                MarkerOptions().position(position).title(camera.name)
            )
            var lastClickedMarker: Marker? = null
            googleMap.setOnMarkerClickListener { marker ->
                val selectedCamera = cameras.find { it.name == marker.title }
                selectedCamera?.let { it ->
                    si.removeAllViews()
                    lastClickedMarker?.setIcon(
                        BitmapDescriptorFactory.defaultMarker(
                            BitmapDescriptorFactory.HUE_RED
                        )
                    )
                    marker.setIcon(
                        BitmapDescriptorFactory.defaultMarker(
                            BitmapDescriptorFactory.HUE_GREEN
                        )
                    )
                }
                lastClickedMarker = marker
            }
        }
    }
}
```

KUVA 16. Karttamerkkien ulkonäköön vaikuttavaa koodia

Kokemattomalle ohjelmistokehittäjälle ChatGPT:n kirjastojen käyttöön antamat ohjeet voivat olla liian epätarkkoja varsinkin koodin oikean sijoittamisen suhteen.

Tekoälytyökaluilla on huomattava vaikutus sovelluskehityksessä. Ongelmien korjaaminen muuttuu useissa erilaisissa virhetilanteissa lähes automaattiseksi ja niissäkin tapauksissa, joissa tekoäly ei tarjoa suoraa ratkaisua, se yleensä antaa hyödyllistä tietoa virheilmoituksen sisällöstä. Tekoälyä voi myös käyttää toimivan koodin parantamiseen kysymällä korvaavia ehdotuksia olemassa olevalle koodille ja siten parantaa koodin laatua.

5.3.5 Kehotteiden kehittäminen ja kehotemuotoilu

Tekoälyn hyödyntämisessä ohjelmoinnin apuna käyttäjän osaamisella on merkitystä tekoälyn tuottamien vastauksien tulkitsemisen lisäksi sopivien kehotteiden kirjoittamisessa. Kehotteiden kehittäminen (55; 56) on osa tekoälyn hyödyntämistä ja se on varsinkin ohjelmointia luovemmilla alueilla, kuten esimerkiksi kuvien ja tekstin tuottamisessa, tärkeä osa hyvien tuloksien saavuttamisessa.

Ohjelmointia tekoälyn avustuksella tehtäessä ei tarvita monimutkaisia luonnollisella kielellä kirjoitettuja kehotteita vaan hyväksyttävään lopputulokseen päästään hyvin pelkistetyllä syötteellä, johon on listattu vain tarvittavat asiat. Esimerkiksi tämän työn eräässä vaiheessa tehdyn kyselyn ensimmäinen versio ”graphql query kotlin” sisältää käytännössä kaiken tarvittavan ohjelmointikielstä rajapintaan sekä sen, mitä rajapintaa käyttäen halutaan tehdä. Syötettä jouduttiin muokkamaan monimutkaisemmaksi sen takia, että vastaukselle oli asetettu tiettyjä ennakkovaatimuksia esimerkiksi kirjastoihin liittyen.

Onko kehote hyvä vai huono riippuu ainakin ohjelmoinnin kaltaisissa teknisissä sovelluskohteissa enimmäkseen siitä, miten hyvän vastauksen se antaa. Yhtenä laatuksena kehotteelle voidaan pitää esimerkiksi sitä, kuinka yhdenmukaisia vastauksia se antaa ja siihen päästään yleensä sillä, että kysymyksessä on tarpeeksi tietoa siitä, mitä halutaan ja miten. Kehotemuotoilua voidaan tehdä esimerkiksi antamalla tekoälylle selkeät ja yksityiskohtaiset ohjeet ja kehittää niitä vaiheittain yrityksen ja erehdyksen kautta.

5.3.6 Ongelmatilanteet tekoälytyökalujen kanssa

Projektia aikana kohdattiin myös joitain haasteita ja ongelmatilanteita tekoälytyökalujen kanssa. Yksi harvinaisista ongelmista oli, että ChatGPT tuotti toimimatonta koodia ja siihen korjausta pyydetessä jäi keskustelu tekoälyn kanssa silmukkaan, jossa tekoäly toistaa samoja vastauksia loputtomasti. Viallisen koodin sisältämään viestiin vastattiin kehotteella, missä kerrottiin koodissa olevasta virheestä, ja Bing Chat vastasi pyytämällä anteeksi virhettään ja tulostamalla täsmälleen saman koodin uudelleen. Yleensä tämän kaltaiset ongelmat ovat kuitenkin käyttäjän virheitä, kun muutos koodissa on niin pieni, että se jää käyttäjältä huomaamatta.

Yleisimpiä ongelmia olivat vanhentuneiden, käytöstä poistuneiden tai muista yhteensopivuusongelmista käytössä olevien ohjelmisto- ja muiden versioiden kanssa kärsivien kirjastojen, termien tai syntaksien sisällyttäminen vastauksiin. Nämä ongelmat olivat kuitenkin yleensä kohtuudella korjattavissa antamalla ChatGPT:lle Android Studion antamat virheilmoitukset.

6 POHDINTA

Tämän opinnäytetyön tavoitteena oli selvittää, mitä hyötyjä ja haasteita tekoälyn käytöllä on ohjelmoinnin apuna. Työssä tarkasteltiin tekoälyn historiaa, erilaisia tekoälytekniikoita, tekoälyn integroimista osaksi sovelluksia sekä tekoälyn eettisiä ja teknisiä kysymyksiä. Työssä toteutettiin myös Android-sovellus, joka hyödynsi tekoälytyökaluja ohjelmoinnissa. Työssä keskityttiin Android-sovelluskehitykseen, mutta työn tuloksia voidaan soveltaa myös muihin ohjelmointiympäristöihin.

Työn tulokset osoittavat, että tekoäly voi olla erittäin tehokas ja hyödyllinen työkalu ohjelmoinnissa, kun sitä käytetään oikein ja vastuullisesti. Tekoäly voi auttaa ohjelmoijaa ratkaisemaan ohjelmointiongelmia nopeammin ja helpommin kuin perinteisesti käytetyt hakukoneet ja muut lähteet. Tekoäly voi tuottaa pitkiä ja monimutkaisia koodeja yksinkertaisilla komennoilla, jos ohjelmoija osaa hyödyntää koodieditorin tekoälylaajennuksia tai muita tekoälytyökaluja.

Tekoäly ei kuitenkaan voi korvata ohjelmoijan omaa osaamista, vaan se on vain yksi työkalu muiden joukossa. Yksinkertaisen kehotteen tuottaman geneerisen koodin käyttökelpoisuus voi olla hyvin rajattua, jos käyttäjältä puuttuu kyky muokata se sopivaksi. Osaamisesta on hyötyä vastauksien ymmärtämisen lisäksi myös kehoitteiden tuottamisessa. Asiantuntijuus mahdollistaa sopivan kehotteen kirjoittamisen niissäkin tilanteissa, jotka vaativat toimiakseen tarkoin määrätyn vastauksen. Hyvä kehote voi olla lista tarvittavista asiasanoista tai ihmismäinen teksti, joka sopisi esimerkiksi keskustelufoorumien käyttäjiltä asiasta kysymiseen.

Opinnäytetyön tulokset osoittavat tekoälyn olevan toimiva apuväline ohjelmoijalle kokemuksen määrästä riippumatta. Tekoälyn käytössä kannattaa noudattaa samoja sääntöjä kuin perinteisessä ohjelmointiin liittyvässä tiedonhaussa, kuten esimerkiksi se, että ymmärtää koodin toiminnan ennen sen käyttöä. Parhaan lopputuloksen saavuttaminen edellyttää, että ohjelmoija tiedostaa tekoälyn toimintaperiaatteet, rajoitukset, virheet ja vaikutukset sekä noudattaa eettisiä ja laillisia sääntöjä tekoälyn käytössä.

LÄHTEET

1. IBM 2023. What is artificial intelligence (AI)? Hakupäivä 17.12.2023. <https://www.ibm.com/topics/artificial-intelligence> (arkistoitu 29.12.2023. <https://archive.is/jxbwG>).
2. IBM 2023. What is strong AI? Hakupäivä 19.12.2023. <https://www.ibm.com/topics/strong-ai> (arkistoitu 1.1.2024. <https://archive.is/0ID2k>).
3. IBM 2023. What is machine learning? Hakupäivä 20.12.2023. <https://www.ibm.com/topics/machine-learning> (arkistoitu 1.1.2024. <https://archive.is/JoQfq>).
4. Microsoft 2023. What is machine learning? Hakupäivä 1.1.2024. <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-machine-learning-platform/> (arkistoitu 20.12.2023. <https://archive.is/wpVL1>).
5. IBM 2023. What is deep learning? Hakupäivä 20.12.2023. <https://www.ibm.com/topics/deep-learning> (arkistoitu 1.1.2024. <https://archive.is/mjsEG>).
6. DeepAI 2023. Deep Learning. Hakupäivä 20.12.2023. <https://deepai.org/machine-learning-glossary-and-terms/deep-learning> (arkistoitu 1.1.2024. <https://archive.is/zR3NH>).
7. IBM 2023. What is natural language processing (NLP)? Hakupäivä 21.12.2023. <https://www.ibm.com/topics/natural-language-processing> (arkistoitu 1.1.2024. <https://archive.is/TdEFN>).
8. Oracle 2023. What Is Natural Language Processing (NLP)? Hakupäivä 21.12.2023. <https://www.oracle.com/ph/artificial-intelligence/what-is-natural-language-processing/> (arkistoitu 1.1.2024. <https://archive.is/Bkadh>).
9. Baeldung 2023. What Are Expert Systems? Hakupäivä 22.12.2023. <https://www.baeldung.com/cs/expert-systems> (arkistoitu 1.1.2024. <https://archive.is/czfxT>).
10. Autoblocks 2023. What is an expert system? Hakupäivä 22.12.2023. <https://www.autoblocks.ai/glossary/expert-system> (arkistoitu 1.1.2024. <https://archive.is/T4iWF>).
11. DeepAI 2023. Pattern Recognition. Hakupäivä 20.12.2023. <https://deepai.org/machine-learning-glossary-and-terms/pattern-recognition> (arkistoitu 1.1.2024. <https://archive.is/UDZCh>).
12. IBM 2023. What is data mining? Hakupäivä 22.12.2023. <https://www.ibm.com/topics/data-mining> (arkistoitu 1.1.2024. <https://archive.is/U5cug>).
13. Usersnap 2023. How AI Transforms User Feedback Analysis. Hakupäivä 18.12.2023. <https://usersnap.com/blog/artificial-intelligence-user-feedback/> (arkistoitu 11.1.2024. <https://archive.is/kQFCY>).

14. IBM 2024. Modernization and Code Generation. Hakupäivä 25.1.2024 <https://www.ibm.com/architectures/hybrid/genai-modernization-and-code-generation> (arkistoitu 25.1.2024. <https://archive.is/k5VN5>).
15. TestingXperts 2021. AI in Software Testing – Benefits, Approaches, Tools to Look in 2023. Hakupäivä 11.1.2024. <https://www.testingxperts.com/blog/AI-in-Software-Testing> (arkistoitu 11.1.2024. <https://archive.is/Uhoxs>).
16. IBM 2023. AI in marketing: How to leverage this powerful new technology for your next campaign. Hakupäivä 22.12.2023. <https://www.ibm.com/blog/ai-in-marketing/> (arkistoitu 11.1.2024. <https://archive.is/emVIT>).
17. Google 2023. Application fundamentals. Hakupäivä 15.7.2023. <https://developer.android.com/guide/components/fundamentals> (arkistoitu 11.1.2024. <https://archive.is/2gqCs>).
18. Google 2023. Meet Android Studio. Hakupäivä 15.7.2023. <https://developer.android.com/studio/intro> (arkistoitu 12.10.2023. <https://archive.is/joW2s>).
19. Gradle Inc. 2024. Gradle Build Tool. Hakupäivä 16.1.2024. <https://gradle.org/> (arkistoitu 16.1.2024. <https://archive.is/B2RUB>).
20. Google 2023. Firebase. Hakupäivä 16.1.2024. <https://firebase.google.com/> (arkistoitu 16.1.2024. <https://archive.is/zsN42>).
21. GitHub 2024. GitHub. Hakupäivä 16.1.2024. <https://github.com/> (arkistoitu 16.1.2024. <https://archive.is/NA5TF>).
22. Git 2023. Getting Started - What is Git? Hakupäivä 16.1.2024. <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F> (arkistoitu 16.1.2024. <https://archive.is/9NlcA>).
23. Google 2024. Espresso. Hakupäivä 16.1.2024. <https://developer.android.com/training/testing/espresso/> (arkistoitu 16.1.2024. <https://archive.is/g6lhr>).
24. Google 2024. Write automated tests with UI Automator. Hakupäivä 16.1.2024. <https://developer.android.com/training/testing/other-components/ui-automator> (arkistoitu 16.1.2024. <https://archive.is/DiXAm>).
25. Google 2023. Design & Plan. Hakupäivä 15.07.2023. <https://developer.android.com/design> (arkistoitu 1.1.2024. <https://archive.is/GKTVO>).
26. Google 2018. Making More with Material. Hakupäivä 15.7.2023. <https://design.google/library/making-more-with-material> (arkistoitu 1.1.2024. <https://archive.is/J1mLw>).
27. Google 2023. Guide to Android app modularization. Hakupäivä 15.7.2023. <https://developer.android.com/topic/modularization> (arkistoitu 16.1.2024. <https://archive.is/wip/KMbtN>).

28. Google 2023. Version control basics. Hakupäivä 15.7.2023. <https://developer.android.com/studio/projects/version-control> (arkistoitu 16.1.2024. <https://archive.is/Xgl81>).
29. Google 2023. Test your app. Hakupäivä 15.7.2023. <https://developer.android.com/studio/test/> (arkistoitu 16.1.2024. <https://archive.is/QxpJM>).
30. Google 2023. Design for Safety. Hakupäivä 15.7.2023. <https://developer.android.com/quality/privacy-and-security> (arkistoitu 16.1.2024. <https://archive.is/humK5>).
31. Google 2024. ML Kit. Hakupäivä 16.1.2024. <https://developers.google.com/ml-kit/> (arkistoitu 16.1.2024. <https://archive.is/UoZal>).
32. TensorFlow 2024. TensorFlow Lite. Hakupäivä 16.1.2024. <https://www.tensorflow.org/lite> (arkistoitu 16.1.2024. <https://archive.is/Q6KRx>).
33. Google 2024. Google Lens. Hakupäivä 16.1.2024. <https://lens.google/> (arkistoitu 16.1.2024. <https://archive.is/4z3Bg>).
34. Google 2024. Google Assistant, your own personal Google. Hakupäivä 16.1.2024. <https://assistant.google.com/> (arkistoitu 16.1.2024. <https://archive.is/w2DYI>).
35. Spotify 2024. Spotify. Hakupäivä 16.1.2024. <https://open.spotify.com/> (arkistoitu 16.1.2024. <https://archive.is/mrMAG>).
36. Netflix 2024. Netflix Finland. Hakupäivä 16.1.2024. <https://www.netflix.com/fi-en/> (arkistoitu 16.1.2024. <https://archive.is/fhDjs>).
37. Microsoft 2024. Microsoft SwiftKey. Hakupäivä 16.1.2024. <https://www.microsoft.com/en-us/swiftkey> (arkistoitu 16.1.2024. <https://archive.is/omRll>).
38. Google 2024. Use Live View on Google Maps. Hakupäivä 16.1.2024. <https://support.google.com/maps/answer/9332056> (arkistoitu 16.1.2024. <https://archive.is/X74Nh>).
39. Unesco 2021. Recommendation on the Ethics of Artificial Intelligence. Hakupäivä 27.1.2024. <https://unesdoc.unesco.org/ark:/48223/pf0000381137> (arkistoitu 27.1.2024 <https://archive.is/mXPYq>).
40. Oulunliikenne.fi 2024. Avoin data. Hakupäivä 16.1.2024. <https://wp.ouunliikenne.fi/avoin-data/> (arkistoitu 16.1.2024. <https://archive.is/uo3cA>).
41. Google 2024. About. Hakupäivä 16.1.2024. <https://maps.google.com/maps/about/#/> (arkistoitu 16.1.2024. <https://archive.is/ccvRG>).
42. The GraphQL Foundation 2024. GraphQL. Hakupäivä 16.1.2024. <https://graphql.org/> (arkistoitu 16.1.2024. <https://archive.is/v4VO5>).
43. Apollo Graph Inc. 2024. Streamlining APIs, Databases & Microservices. <https://www.apollo-graphql.com/> (arkistoitu 16.1.2024. <https://archive.is/UhzJ6>).

44. OkHttp 2022. OkHttp: Overview. Block Inc. Hakupäivä 16.1.2024. <https://square.github.io/okhttp/> (arkistoitu 16.1.2024. <https://archive.is/ctNgY>).
45. OpenAI 2024. GPT-4 is OpenAI's most advanced system, producing safer and more useful responses. Hakupäivä 16.1.2024. <https://openai.com/gpt-4> (arkistoitu 16.1.2024. <https://archive.is/TMPVV>).
46. Guiding Tech Media 2023. GPT-4 Details Have Been Leaked! Hakupäivä 16.1.2024. <https://www.kdnuggets.com/2023/07/gpt4-details-leaked.html> (arkistoitu 16.1.2024. <https://archive.is/FVaaH>).
47. OpenAI 2024. GPT-4. Hakupäivä 16.1.2024. <https://openai.com/research/gpt-4> (arkistoitu 16.1.2024. <https://archive.is/6IBUJ>).
48. Microsoft 2024. Bing. Hakupäivä 16.1.2024. <https://www.bing.com/chat> (arkistoitu 16.1.2024. <https://archive.is/JPW9>).
49. Microsoft 2024. Copilot with GPT-4. Hakupäivä 16.1.2024. <https://www.bing.com/chat> (arkistoitu 16.1.2024. <https://archive.is/DYFuN>).
50. OpenAI 2024. ChatGPT. Hakupäivä 16.1.2024. <https://openai.com/chatgpt> (arkistoitu 16.1.2024. <https://archive.is/Ty3l0>).
51. OpenAI 2024. ChatGPT Pricing. Hakupäivä 16.1.2024. <https://openai.com/chatgpt/pricing> (arkistoitu 16.1.2024. <https://archive.is/5w4EO>).
52. Google 2024. Download Android Studio & App Tools. Hakupäivä 16.1.2024. <https://developer.android.com/studio> (arkistoitu 16.1.2024. <https://archive.is/00Tq1>).
53. Microsoft 2024. Visual Studio Code. Hakupäivä 16.1.2024. <https://code.visualstudio.com/> (arkistoitu 16.1.2024. <https://archive.is/cuayE>).
54. GitHub, Inc. 2024. GitHub Copilot. Hakupäivä 16.1.2024. <https://github.com/features/copilot> (arkistoitu 16.1.2024. <https://archive.is/PF561>).
55. IBM 2024. What is prompt engineering? Hakupäivä 27.1.2024. <https://www.ibm.com/topics/prompt-engineering> (arkistoitu 27.1.2024. <https://archive.is/pn28B>).
56. Google 2024. Prompt Engineering for Generative AI. Hakupäivä 27.1.2024. <https://developers.google.com/machine-learning/resources/prompt-eng> (arkistoitu 27.1.2024. <https://archive.is/BBgcz>).