Suurten kielimallien hyödyntäminen frontend-kehityksessä

Turun yliopisto
Tietotekniikan laitos
TkK-tutkielma
Tietotekniikka
Toukokuu 2025
Santeri Maula

TURUN YLIOPISTO
Tietotekniikan laitos

Santeri Maula: Suurten kielimallien hyödyntäminen frontend-kehityksessä

TkK-tutkielma, 24 s.
Tietotekniikka
Toukokuu 2025

Tekoälyllä on ollut valtava vaikutus ohjelmistokehityksen käytäntöihin, koulutukseen ja tekoälyn eettiseen soveltamiseen ohjelmoinnin tukena. Tämän kandidaatintutkielman tavoitteena on lisätä ymmärrystä siitä, miten suuria kielimalleja voidaan hyödyntää frontend-kehityksessä, ja millaisia vaikutuksia niillä on ohjelmistokehitysprosesseihin. Tutkielma tarkastelee tekoälyavusteisen kehitystyön mahdollisuuksia parantaa ohjelmistojen laatua ja kehitystehokkuutta nopeasti kehittyvällä alalla. Tutkimusmenetelmänä käytettiin kirjallisuuskatsausta, jonka aineistona hyödynnettiin viimeisen viiden vuoden aikana julkaistuja tieteellisiä artikkeleita, konferenssijulkaisuja sekä tutkimusraportteja. Aineisto analysoitiin laadullisesti keskittyen suuriin kielimalleihin ja niiden sovelluksiin frontend-ohjelmoinnissa. Tutkielman perusteella suuret kielimallit kykenevät nopeuttamaan käyttöliittymien toteutusta, tukea suunnittelua ja automatisoida testausta, mutta niiden hyödyntäminen vaatii kehittäjältä edelleen teknistä osaamista ja kriittistä ajattelua.

Asiasanat: Frontend, ChatGPT, UI/UX, Suuret kielimallit

Sisällys

1 Johdanto

Generatiivinen tekoäly (engl. generative artificial intelligence, GenAI) on malli, joka pystyy tuottamaan uutta sisältöä, kuten tekstiä kuvia, ääntä tai ohjelmakoodia [1]. Viime vuosina tekoäly on mullistanut ohjelmoinnin ja ohjelmistokehityksen. Generatiiviset suuret kielimallit, kuten OpenAI:n vuonna 2022 julkaisema ChatGPT [2] ja GitHubin vuonna 2021 julkaisema Copilot [3], ovat muuttaneet tapaa, jolla ohjelmistoja suunnitellaan, kehitetään ja käytetään. Näiden työkalujen yleistyminen on tuonut ohjelmistokehitykseen uudenlaisen vuorovaikutusmallin, jossa kehittäjä saa tekoälyltä tukea esimerkiksi ohjelmakoodin kirjoittamisessa, virheiden löytämisessä tai korjaamisessa sekä dokumentaation kirjoittamisessa.

Frontend-kehityksessä, eli verkkosovellusten käyttöliittymien ja selaimen puolella toimivan logiikan suunnittelussa ja toteutuksessa, generatiivisilla malleilla on merkittävä potentiaali. Malleilla voidaan mahdollisesti nopeuttaa kehitysprosessia ja työnkulkua sekä parantaa tuottavuutta. Esimerkkejä generatiivisten mallien käyttökohteista frontend-kehityksessä ovat ohjelmakoodin generointi ja selittäminen, virheiden korjaaminen ja löytäminen, suunnittelu sekä dokumentaation generointi.

Tutkielman tarkoituksena on selvittää, miten generatiivisia malleja voidaan hyödyntää frontend-kehityksen tukena ja mitä vaikutuksia sillä on kehityksen prosesseihin. Tavoitteena on lisätä ymmärrystä siitä, millaisia käyttökohteita malleilla on käyttöliittymien suunnittelussa ja toteutuksessa, ja millä tavoin ne voivat tukea kehittäjiä tehtävissä. Lisäksi tutkielmassa tarkastellaan mallien tuomia etuja, haastei-

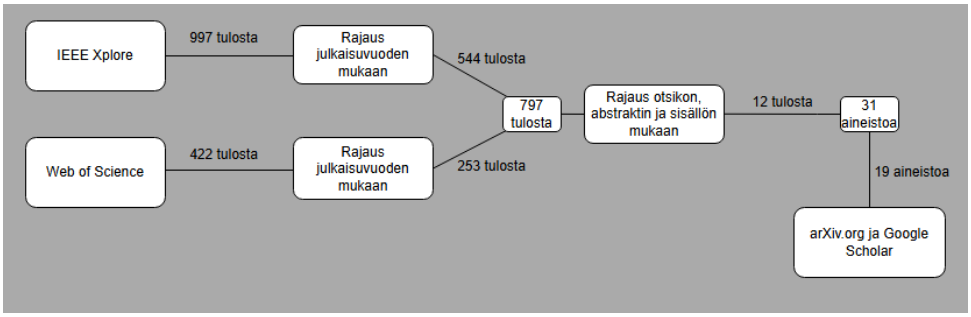
ta ja rajoituksia kuten ohjelmakoodin tarkkuutta ja luotettavuutta. Näiden tavoitteiden pohjalta tutkielmassa tarkastellaan seuraavia tutkimuskysymyksiä:

TK1 Miten suuria kielimalleja voidaan hyödyntää frontend-kehityksen apuvälineinä?

TK2 Millaisia etuja tai haasteita suurten kielimallien käyttö tuo frontend-kehitykseen?

TK3 Kuinka tarkkoja ja luotettavia suurten kielimallien tuottamat ohjelmointiratkaisut ovat?

Tutkielman tutkimusmenetelmä on kirjallisuuskatsaus, jonka tavoitteena on koota ja analysoida ajankohtaista tutkimustietoa suurien kielimallien hyödyntämisestä frontend-kehityksessä. Tutkielman tiedonhaku toteutettiin kahdessa vaiheessa. Ensimmäisessä vaiheessa etsittiin tutkimuskysymyksiin vastaavaa tieteellistä aineistoa generatiivisten mallien soveltamisesta frontend-kehityksessä. Hakua varten laadittiin seuraava hakulauseke: *(generative OR "generative AI"OR "large language model"OR "artificial intelligence") AND ("frontend development"OR "user interface"OR "UI development"OR "web development") AND (tool* OR assist*)*. Hakulausekkeella etsittiin aineistoa IEEE Xplore- ja Web of Science-tietokannoista. Hakutulokset rajattiin viimeisen viiden vuoden aikana julkaistuihin teoksiin ajankohtaisen tutkimustiedon varmistamiseksi. Toisessa vaiheessa kerättiin taustakirjallisuutta tekoälyn menetelmistä ja suurten kielimallien toimintaperiaatteista, jotta saataisiin teoreettinen viitekehys suurille kielimalleille. Näihin etsintöihin käytettiin arXiv.org-julkaisuarkistoa sekä Google Scholaria, joiden avulla haettiin ajankohtaisia tutkimus- ja konferenssiartikkeleita. Kuvassa 1.1 esitetään hakuprosessin eteneminen ja hakutulosten rajaukset.



Kuva 1.1: Hakuprosessin eteneminen ja aineiston valinnan rajaukset.

Tutkielman rakenne koostuu johdannon lisäksi viidestä luvusta. Toisessa luvussa käsitellään tekoälyn keskeisiä menetelmiä, erityisesti generatiivisiin malleihin liittyviä teknologioita, joita tarvitaan aiheen ymmärtämiseksi. Kolmannessa luvussa tarkastellaan konkreettisia käyttötapauksia ja sovelluksia generatiivisten mallien käytöstä ohjelmoinnissa yleisesti. Neljännessä luvussa tutkitaan ja esitellään mallien käyttötapauksia frontend-kehityksessä. Viidennessä luvussa pohditaan mallien etuja, haasteita sekä suurten kielimallien luotettavuutta ja tarkkuutta frontend-kehityksessä. Kuudes luku on yhteenveto, jossa esitetään johtopäätökset sekä vastataan tutkimuskysymyksiin.

2 Tekoälyn menetelmät

Tässä luvussa käsitellään tekoälyn menetelmiä, kuten syväoppimista, koneoppimista, luonnollisen kielen prosessointia, kielimalleja sekä suuria kielimalleja. Kone- ja syväoppiminen ovat teoreettinen pohja suurille kielimalleille, mutta niiden yksityiskohtainen käsittely ei ole välttämätöntä tutkielman kannalta. Sen sijaan luonnollisen kielen prosessointi, kielimallit sekä suuret kielimallit ovat suoraan yhteydessä sovelluksiin, joissa generatiivisia malleja voidaan hyödyntää frontend-kehityksessä.

2.1 Koneoppiminen ja syväoppiminen

Koneoppimisella (engl. machine learning) tarkoitetaan tekoälyn menetelmää, jossa tietokoneet oppivat tunnistamaan kuvioita ja tekemään päätöksiä ilman, että tietokoneelle tarvitsisi ohjelmoida jokaista tehtävää erikseen. Koneoppiminen jaetaan yleisesti kolmeen kategoriaan: **Ohjattuun oppimiseen (engl. supervised learning)**, **ohjaamattomaan oppimiseen (engl. supervised learning)** sekä **vahvistusoppimiseen (engl. reinforcement learning)**. Ohjatussa oppimisessa koneelle annetaan valittu syöte ja haluttu ratkaisu. Ohjaamattomassa oppimisessa koneelle ei anneta vastausta syötteeseen, vaan se päättelee vastauksen opetusdatassa olevien säännönmukaisuuksien perusteella. Vahvistusoppimisessa kone saa palautetta sen tuottamien vastauksien onnistuneisuudesta. [4]

Syväoppiminen (engl. deep learning) on koneoppimisen osa-alue, jossa käytetään monikerroksisia neuroverkkoja. Syväoppimisessa verkon eri kerrokset tunnistavat eri

tasoilla dataan liittyviä ominaisuuksia, kuten reunaviivoja kuvissa tai lauseenrakenteita teksteissä. Koneoppimisen ongelmana oli pitkään suurien datamäärien prosessointi. Syväoppiminen ratkaisi tämän ongelman, sillä se pystyy operoimaan suurilla datamäärillä.

Neuroverkot koostuvat tyypillisesti kolmesta pääkerroksesta: **syötekerroksesta** (engl. input layer), **piilokerroksesta** (engl. hidden layers) ja **ulostulokerroksesta** (engl. output layer). Syväoppimisessa käytettävät neuroverkkoarkkitehtuurit voidaan jakaa eri tyyppeihin sen mukaan, minkälaista dataa ne käsittelevät. Näistä esimerkkinä ovat rekursiiviset neuroverkot (engl. recursive neural network, RNN) ja konvoluutioneuroverkot (engl. convolutional neural network, CNN). RNN-malleja käytetään sekventiaalisen datan, kuten tekstin tai puheen käsittelyyn hyödyntämällä takaisinkytkentää (engl. backpropagation). Takaisinkytkentä mahdollistaa aikaisempien tilojen hyödyntämisen myöhemmissä vaiheissa. [5] CNN-malleja käytetään erityisesti kuvankäsittelyssä ja luokittelutehtävissä. CNN-malli hyödyntää paikallisia piirteitä ja vähentää laskennallista monimutkaisuutta käyttämällä kerroksellista hierarkiaa kuvasta saatavien ominaisuuksien erotteluun. [4]

2.2 Luonnollisen kielen prosessointi

Luonnollisen kielen prosessointi (engl. natural language processing, NLP) on tekoälyn osa-alue, jossa prosessoidaan ja analysoidaan puhuttua tai kirjoitettua kieltä. NLP:n tavoite on saada tietokoneet ymmärtämään ihmiskielen lauseita tai sanoja. NLP voidaan jakaa kahteen pääkomponenttiin: **Luonnollisen kielen ymmärtäminen (engl. natural language understanding, NLU)** ja **luonnollisen kielen generointi (eng. natural language generation, NLG)**. NLU pyrkii ymmärtämään ihmiskielen merkitystä ja rakennetta. NLU sisältää eri tehtäviä, kuten sanaluokkien tunnistamista, syntaktista analyysia ja sementtistä tulkintaa. NLG pyrkii tuottamaan tekstiä ymmärrettävässä ja loogisessa muodossa. NLG:n tehtäviä ovat

sisällön valinta, tekstin rakenteen suunnittelu ja sanaston optimointi. [6]

Yksi NLP:n ensimmäisistä vaiheista on tokenisaatio, jossa tekstimuotoinen data muunnetaan tokeneiksi eli yksiköiksi, joita syväoppimismallit voivat sitten prosessoida. Tokenit voivat olla sanoja, lauseita tai yksittäisiä merkkejä, riippuen analyysin tarpeista. Kun teksti on tokenisoitu, se on muunnettava numeeriseen muotoon, sillä raakateksti ei sellaisenaan sovellu koneelliseen käsittelyyn. Numeerisessa muodossa sanat tai lauseet esitetään vektorimuodossa. Näin mahdollistetaan kielimalleille semanttinen analyysi ja kontekstin huomiointi. [7]

Ennen syväoppimisen yleistymistä tekstidataa esitettiin tilastollisin menetelmin. Näitä menetelmiä olivat esimerkiksi One-hot encoding, Bag-of-Words (BoW) ja TF-IDF (term frequency-inverse document frequency). Näiden menetelmien haasteena oli, että sanojen merkityssuhteita ei huomioitu. Tämän vuoksi kehitettiin sanavektoreita hyödyntävät neuroverkkopohjaiset mallit, kuten Word2Vec [8] ja GloVe (Global Vectors for Word Representation) [9]. Vaikka Word2Vec ja GloVe paransivat merkittävästi sanojen merkityksen mallintamista, ne käyttivät edelleen staattista sanavektoriesitystä, jossa jokaiselle sanalle määritettiin vain yksi vektorimuotoinen esitys. Yksittäisessä vektorimuotoisessa esityksessä monimerkityksiset sanat saavat aina saman vektoriesityksen riippumatta kontekstista. Esimerkiksi englannin kielen sana 'spring' voi tarkoittaa suomennettuna kevättä, vesilähdettä, jousta jne. Tähän on kehitetty staattisten sanavektoriesitysten korvaamat transformer-arkkitehtuuria käyttävät syväoppimismallit. Nämä mallit hyödyntävät monikerroksisia neuroverkkoja, joiden avulla pystytään muuntamaan dynaamisesti sanojen merkityksiä niiden kontekstin perusteella. [10]

2.3 Kielimallit

Transformer-arkkitehtuuri on yksi merkittävimmistä edistyksistä NLP:n ja syväoppimisen aloilla. Transformerit koostuvat pääasiassa kahdesta osasta: enkooderista

(engl. encoder) ja dekooderista (engl. decoder). Enkooderi muuntaa tekstidatan kontekstuaaliseksi vektoriesitykseksi, ja dekooderi käyttää tätä esitystä luodakseen kohdeulostulon. Transformer-arkkitehtuurin keskeisin mekanismi on itsehuomio (engl. self-attention), jonka vuoksi on mahdollista mallintaa sanojen välistä yhteyttä samanaikaisesti koko syötteessä. Toisin kuin esimerkiksi RNN-mallit, jotka käsittelevät tekstisyötettä järjestyksessä sana sanalta, transformer-arkkitehtuuria käyttävät mallit kykenevät ymmärtämään sanojen välistä kontekstia ja niiden riippuvuuksia. Itsehuomion avulla mallit voivat siis huomioida sanaa ympäröivän lauseen kokonaisuudessaan. [11]

Kielimallilla (engl. language model, LM) kuvataan tekoälymallia, joka on suunniteltu analysoimaan, ymmärtämään ja tuottamaan luonnollista kieltä. Kielimallien keskeinen tehtävä on ennustaa seuraava sana tai merkkijono annetussa tekstikonteksissa. Kielimallit ovat ajan saatossa kehittyneet tilastollisia menetelmiä käyttävistä malleista neuroverkkoja käyttäviin malleihin. Näistä siirryttiin esikoulutusta käyttäviin kielimalleihin (engl. pre-trained language model, PLM), ja näistä nykyään suuriin kielimalleihin. PLM:llä tarkoitetaan kielimallia, joka on koulutettu suurella datamäärällä ennen sen käyttöä eri tehtäviin. Näiden mallien koulutuksen tavoitteena on, että malli oppii yleisen kielen esityksen, jota voidaan sittemmin käyttää eri NLP-tehtävissä, kuten tekstin luokittelussa. Esikoulutusvaiheiden jälkeen mallit voidaan hienosäätää (engl. fine-tuning) eri tehtäviin pienemmillä, tehtäväkohtaisilla aineistoilla. [12]

Suuret kielimallit (engl. large language models, LLM) ovat esikoulutettuja malleja, jotka eroavat perinteisistä malleista kooltaan ja laskennalliselta vaatimustasoltaan. Esimerkkejä transformer-arkkitehtuuria käyttävistä suurista kielimalleista ovat Googlen kehittämä BERT (Bidirectional Encoder Representations from Transformers) [13] ja OpenAI:n kehittämä GPT (Generative Pretrained Transformer). Suuria kielimalleja voidaan luokitella niiden arkkitehtuurin ja koulutustavan perusteella. Ne

jakautuvat kolmeen luokkaan: **dekooderipohjaisiin malleihin** (engl. decoder-only, esim. GPT), **enkooderipohjaisiin malleihin** (engl. encoder-only, esim. BERT), **enkooderi-dekooderi -pohjaisiin malleihin** (engl. encoder-decoder, esim. T5). Dekooderipohjaiset mallit toimivat autoregressiivisesti, eli ne ennustavat seuraavan sanan annetun kontekstin perusteella. Enkooderipohjaiset mallit käyttävät peitetyn kielen mallinnusta (engl. masked language modeling, MLM), jossa osa sanoista piilotetaan ja malli oppii ennustamaan piilotetut sanat niitä ympäröivän kontekstin perusteella. Enkooderi-dekooderi-mallit yhdistävät enkooderi- ja dekooderipohjaisten mallinnuksien lähestymistavat. Ne soveltuvat generatiivisiin tehtäviin, kuten tiivistämiseen ja konekääntämiseen. [14]

Suurten kielimallien toiminta perustuu valtavaan määrään parametreja. Parametrit ovat mallien oppimisessa keskeisiä. Parametrit ovat neuroverkon painokertoimia, jotka säätävät, millä tavalla malli käsittelee ja tulkitsee tietoa. Parametrien määrä vaihtelee malleista riippuen miljardeista jopa biljooniin. [15]

2.4 Nykyaikaset suuret kielimallit

Suurten kielimallien nopea kehitys on mahdollistanut mallien hyödyntämisen eri tehtäviin. Mallit voidaan jakaa eri kategorioihin kyvykkyyden ja käyttötapojen mukaan. Luvussa 2.3 käsitellyt perinteiset suuret kielimallit ovat generatiivisia malleja, jotka prosessoivat tekstipohjaista dataa ja kykenevät tuottamaan kielellisesti luonnollista ja johdonmukaista tekstiä. Mallien koulutuksessa käytetään valtavia määriä tekstidataa, minkä ansiosta mallit suoriutuvat hyvin luonnolliseen kieleen liittyvissä tehtävissä, kuten käännöksissä, tiivistämisessä ja kysymys-vastaustehtävissä. Kasner ja Dušek osoittivat tutkimuksellaan, että vaikka mallit pystyvät tuottamaan sujuvaa ja johdonmukaista tekstiä, niillä on kuitenkin vielä merkittäviä haasteita semanttisessa tarkkuudessa. [16]

Multimodaaliset suuret kielimallit (engl. multimodal large language model, MM-

LLM) ovat laajennus perinteisistä suurista kielimalleista. Ne on suunniteltu käsittelemään useita syötemuotoja, kuten tekstiä, kuvia, videoita, ääntä ja 3D-dataa. MM-LLM:ssä perinteinen suuri kielimalli toimii pohjalla kognitiivisena moottorina, jota laajennetaan multimodaalisuuteen erikoistuneilla komponenteilla, kuten modaliteettikoodereilla ja projektoreilla, jotta eri modaalisuudet voidaan yhdistää yhteen malliin. MM-LLM:t säilyttävät suurien kielimallien kyvyt, mutta laajentavat ne multimodaalisiin tehtäviin. Multimodaalisuudella mahdollistetaan tehtävät, joissa yhdistetään esimerkiksi kuvan tulkintaa ja tekstin tuottamista tai puheen ja tekstin vuorovaikutusta. [17]

Päättelymallit (engl. reasoning models) ovat generatiivisia malleja, jotka on suunniteltu ratkaisemaan loogista ajattelua vaativia tehtäviä. Päättelymallit eroavat perinteisistä suurista kielimalleista siten, että ne pyrkivät mallintamaan ajatteluprosessia tehtävän ratkaisemiseksi. OpenAI:n o1-malli on esimerkki päättelymallista. Sen sijaan, että mallin kokoa tai koulutusdataa kasvatettaisiin, sen kehityksessä on hyödynnetty Test-time Compute -menetelmiä, joiden avulla malli käyttää enemmän aikaa ja resursseja ratkaisun harkintaan ennen vastaamista. Wu ym. havaitsivat tutkimuksessaan, että o1-malli saavutti selvästi parempia tuloksia useilla päättelyä vaativilla alueilla, kuten matematiikassa ja ohjelmoinnissa kuin muut vastaavat mallit. Malli hyödyntää useita päättelymenetelmiä, kuten hajota ja hallitse-algoritmia, itsekritiikkiä ja parantamista (self-refinement), sekä systemaattista analyysia. Tutkimuksessa kävi ilmi, että o1 osaa mukauttaa päättelytyyliään tehtävän mukaan: yleistietotehtävissä malli painottaa kontekstin ymmärtämistä, kun taas teknisissä ongelmissa se soveltaa algoritmeja. [18]

Autonomiset agentit (engl. autonomous agents) ovat järjestelmiä, jotka pystyvät havainnoimaan ympäristöään, tekemään päätöksiä ja suorittamaan tehtäviä itsenäisesti. Ne käyttävät suurten kielimallien luonnollisen kielen ymmärrystä suunnitteluun, muistiin ja toimintaan. Suuria kielimalleja hyödyntävät agentit rakentu-

vat tyypillisesti neljästä moduulista: profiili, muisti, suunnittelu ja toiminta. Moduulien avulla agentit oppivat kokemuksistaan, reflektoivat aiempaa toimintaansa ja sopeuttaa käyttäytymistään dynaamisissa ympäristöissä. Toisin kuin perinteiset agenttijärjestelmät, LLM-pohjaiset agentit hyödyntävät laajaa taustatietoa, suunnittelevat ja suorittavat tehtäviä ilman erillistä koulutusta sekä vuorovaikuttavat ihmisten ja muiden agenttien kanssa luonnollisella kielellä. [19]

Suurten kielimallien koulutuksessa esiintyy myös eri haasteita, kuten hallusinaatiota tai shortcut learning:ia. Hallusinaatiossa malli tuottaa sujuvalta vaikuttavaa tekstiä, joka on kuitenkin sisällöllisesti virheellistä tai opetuslähteestä poikkeavaa. Hallusinaatiota ovat esimerkiksi ristiriitaiset väittämät tai lisätyt tiedot, joita ei voida vahvistaa syöteaineistosta. [20] Shortcut learning tarkoittaa ilmiötä, jossa syväoppivat mallit oppivat ratkaisemaan tehtäviä hyödyntämällä yksinkertaisia ja usein epäolennaisia vihjeitä opetusdatassa sen sijaan, että ne ymmärtäisivät tehtävän varsinaisen sisällön. Esimerkiksi kuvantunnistuksessa tekoäly voi perustaa vastauksensa kuvassa esiintyvään taustaan, eikä itse kuvantunnistuksen kohteeseen, jolloin malli käyttää ns. oikotietä vastauksen saamiseen. [21][22]

3 Suuret kielimallit ohjelmoinnissa

Tässä luvussa käsitellään suurten kielimallien hyödyntämistä ohjelmoinnissa. Luvussa tarkastellaan, millaisiin tehtäviin malleja voidaan soveltaa, miten ne tukevat kehittäjiä eri kehitysvaiheissa ja millaisia työkaluja sekä käyttötapauksia on jo saatavilla. Lisäksi tarkastellaan mallien kyvykkyyttä ohjelmakoodin generoinnissa. Luvun tavoitteena on muodostaa selkeä kuva siitä, miten mallit integroituvat osaksi ohjelmistokehitystä. Aluksi tarkastellaan, kuinka malleja hyödynnetään ohjelmoinnissa yleisesti, jonka jälkeen tarkastellaan mallien kyvykkyyttä tuottaa ohjelmakoodia.

3.1 Suuret kielimallit ohjelmoinnissa

Suuret kielimallit ovat nousemassa merkittäväksi työkaluksi ohjelmistokehityksessä ja sen vaikutukset ulottuvat koko kehitysprosessin läpi suunnittelusta julkaisuun. Suuret kielimallit, kuten ChatGPT ja Copilot, mahdollistavat tehtävien automatisoinnin. Näitä tehtäviä ovat esimerkiksi koodin täydennys, testitapausten luonti, dokumentaatio sekä virheiden tunnistus ja korjaus. Toisin kuin perinteiset hakukoneet ja ohjelmointifoorumit, mallit eivät etsi olemassa olevia vastauksia näistä lähteistä, vaan tuottavat uusia ratkaisuja annetun syötteen sekä koulutusdatan perusteella. Tällöin saadaan nopeammin ohjelmakoodille esimerkkejä, selityksiä tai vaihtoehtoisia toteutustapoja. [23]

3.1.1 Ohjelmakoodin generointi

Suuret kielimallit, joita hyödynnetään ohjelmakoodin generoinnissa, koulutetaan valtavilla määrillä lähdekoodia eri ohjelmointikielistä. Nämä mallit oppivat tilastollisia yhteyksiä sanojen, lauseiden ja ohjelmakoodirakenteiden välillä, jonka ansiosta ne pystyvät ennustamaan ja tuottamaan loogisesti jatkuvaa ohjelmakoodia annetun syötteen pohjalta. Syöte voi olla luonnollisen kielen kuvaus, osittainen koodinpätkä tai muu konteksti, johon malli ehdottaa jatkoa, ratkaisua tai toiminnallisuuksia. Ohjelmakoodin generoinnissa malli arvioi, mikä koodin jatko on todennäköisin syötteen perusteella. ChatGPT tuottaa koodia tehtävänannon mukaan ja selittää sen toiminnan, kun taas Copilot integroidaan kehitysympäristöön, jolloin se tarjoaa automaattisesti koodiehdotuksia reaaliaikaisesti. Molemmat mallit tukevat useita ohjelmointikieliä sekä kykenevät avustamaan kuin yksittäisten rivien, myös laajempien rakenteiden luomisessa. [24]

3.1.2 Ohjelmakoodin testaus

Ohjelmistotestaus on tärkeä osa ohjelmistokehitystä. Sen tavoitteena on varmistaa, että ohjelma toimii odotetulla tavalla, se täyttää annetut vaatimukset sekä olisi mahdollisimman virheetön. Testaus voi olla esimerkiksi yksikkötestausta, integraatiotestausta, käyttöliittymätestausta tai suorituskyvyn arviointia. Perinteisesti testien suunnittelu, kirjoittaminen ja suoritus ovat olleet manuaalisia vaiheita, mutta suuret kielimallit ovat mahdollistaneet testien automatisoinnin.

Suurilla kielimalleilla, kuten ChatGPT:llä, on kyky ymmärtää luonnollista kieltä ja käsitellä monimuotoisia tekstipohjaisia lähteitä, kuten commit-viestejä ja dokumentaatioita. Näin mallit kykenevät automatisoimaan testitapaukset, testidatat sekä odotettujen tuloksien tuottamiset. LLM-pohjaiset mallit oppivat laajan opetusdatansa, kuten ohjelmakoodien, dokumentaatioiden ja keskustelufoorumien, pohjalta käyttäjien todennäköisimpiä toimintamalleja ohjelmistojärjestelmissä. Tämän vuok-

si mallit pystyvät luomaan testitapauksia, jotka mallintavat esimerkiksi mahdollisia erikoistapauksia (engl. edge condition) ilman erillistä sääntöpohjaista ohjelmointia. [25]

3.1.3 Ohjelmakoodin dokumentointi

Dokumentoinnin tarkoituksena on kuvata ohjelmakoodin toiminnallisuutta, tarkoitusta ja käyttöä. Suuret kielimallit ovat tuoneet tehokkaan tavan tuottaa dokumentaatiota automaattisesti. Koodin dokumentointi voidaan nähdä aikaa vievänä sekä tylsänä tehtävänä, jolloin sen automatisointi voi tehostaa kehitysprosesseja ja parantaa ohjelmistojen ylläpidettävyyttä.

Dvivedi ym. [26] havaitsivat tutkimuksessaan, että suurten kielimallien kyky tuottaa dokumentaatiota vaihteli niille annetuista tasoista. Tutkimuksessa vertailtiin eri mallien kyvykkyyttä seuraavilla tasoilla: rivitasolla (engl. inline level documentation), funktiotasolla (engl. function level documentation), luokkatasolla (engl. class level documentation) ja kansiotasolla (engl. folder level documentation). Rivitasolla mallien tehtävä oli dokumentoida yksittäisiä ohjelmakoodirivejä ja funktiotasolla yksittäisiä funktioita (toiminta, parametrit ja paluuarvot). Tiedostotasolla mallien tehtävä oli dokumentoida tiedoston koko rakennetta sekä tarkoitusta. Kansiotasolla malleille annettiin koko ohjelmistomoduuli tai tiedostoryhmä dokumentoitavaksi tavoitteena esittää, miten useat luokat tai tiedostot liittyvät toisiinsa ja minkä kokonaisuuden ne muodostavat.

Tutkimuksessa havaittiin, että rivi-, funktio- sekä luokkatasolla generatiiviset mallit tuottivat johdonmukaista ja ymmärrettävää dokumentaatiota. Kansiotason dokumentoinnin analyysi jätettiin pois, sillä useimmat tutkimuksessa käytetyt mallit eivät kyenneet käsittelemään useita tiedostoja rinnakkain. OpenAI:n GPT-4 osoitti kuitenkin potentiaalia käsitellä ja jäsentää laajempia kokonaisuuksia, mikä viittaisi siihen, että kansiotason dokumentointi edellyttää kehittyneempiä malleja. [26]

3.1.4 Virheiden etsintä ja korjaus

Virheiden etsiminen ja korjaaminen eli debuggaus on yksi yleisimmistä osista ohjelmistokehityksessä. Etenkin suurissa ja monimutkaisissa järjestelmissä debuggaus on työlästä sekä aikaa vievä prosessi. Suurten kielimallien avulla voidaan mahdollisesti helpottaa ja nopeuttaa virheiden paikantamista ja ymmärtämistä sekä analysoida ohjelmistojen lokitietoja, jotka saattavat olla muuten vaikeasti hahmotettavissa. Mallien vahvuus debuggaamisessa perustuu niiden kykyyn tunnistaa ohjelmoinnissa toistuvia kaavoja, käsitellä luonnollista kieltä sekä yleistää ratkaisumalleja koulutusaineistoista. [27]

Sakib, Khan ja Karim [27] havaitsivat tutkimuksessaan, että ChatGPT kykeni tuottamaan 128:sta sille annetusta ohjelmointitehtävästä 92 oikeaa ratkaisua, joista 84 ratkaistiin ensimmäisellä yrityksellä ja kahdeksan debuggaamisen jälkeen. Tutkimuksessa tehtävät jaettiin vaikeustason mukaan helppoihin, keskivaikeisiin ja vaikeisiin. ChatGPT:llä oli eniten ongelmia vaikeiden tehtävien ratkaisuissa myös debuggaamisen jälkeen. Tulokset viittaavat siihen, että ChatGPT osasi käsitellä hyvin määriteltyjä ja rakenteeltaan selkeitä ongelmia sekä suoriutui virheiden paikantamisesta ja korjauksesta, kun virheet olivat syntaktisia tai perustuivat yleisiin ohjelmointikaavoihin. Vaikeudet käsitellä ongelmia korostuivat tehtävissä, jotka vaativat syvällistä päättelyä tai laajaa kontekstin hallintaa.

3.2 Mallien kyvykkyys tuottaa ohjelmakoodia

Suurten kielimallien, kuten OpenAI:n GPT-3.5:n ja GPT-4:n, käyttö ohjelmointitehtävissä on yleistynyt nopeasti, mutta niiden kyvykkyys koodin tuottamisessa vaatii syystemaattista arviointia. Agarwal ym. [28] kehittivät tutkimuksessaan arviointikehyksen, jolla mitattiin mallien suorituskykyä ohjelmointitehtävissä aidossa kehitysympäristössä. Arviointi keskittyi viiteen ohjelmistokehityksen osa-alueeseen:

dokumentaation generointi, virheiden korjaus, koodin generointi luonnollisesta kielestä, testien generointi sekä ohjelmiston rakenteen ymmärtäminen.

Mallin kyvykkyyttä generoida ohjelmakoodia luonnollisen kielen kuvauksesta mitattiin kahden kriteerin perusteella: ratkaisun syntaktinen oikeellisuus sekä testien läpäisyaste. Mallin tehtävänä oli tuottaa annettuun ohjelmaan funktio pelkän kuvauksen perusteella. Ohjelmasta oli poistettu alkuperäisen funktion koodi ja korvattu se kommentilla, jossa pyydettiin mallia generoimaan koodia funktion sisälle. Tämän jälkeen mallin generoima koodi sijoitettiin takaisin ohjelmaan ja sitä testattiin olemassa olevalla testikannalla. Tavoitteena oli arvioida, miten hyvin koodi täyttää sille asetetut toiminnalliset vaatimukset käytännön kehitystyössä.

Tutkimustulosten perusteella GPT-4 suoriutui malleista parhaiten. GPT-4 tuotti korkealaatuista, syntaktisesti oikeaa ja testit läpäisevää koodia useilla ohjelmointikielillä. GPT-3.5 saavutti lähes vastaavan tason, mutta käsitteli virheitä yksinkertaisemmin ja jäi hieman jälkeen testien läpäisyprosentissa. Vaikka tulokset osoittavat, että mallit kykenevät tuottamaan laadukasta koodia tehokkaasti, tutkimus korosti myös mallien haavoittuvuutta tilanteissa, joissa konteksti oli epäselvä tai suorittamiseen vaadittiin syvällisemmän ohjelmointilogiikan ymmärrystä. Mallien tekemät virheelliset korjaukset tai epäjohdonmukaiset muutokset voivat johtua väärin ymmärretystä tehtävänannosta tai rajoitetusta kontekstista. Lisäksi havaittiin, että GPT-4 saattoi epäonnistua tehtävissä, joissa se yritti tuottaa liian monimutkaista ratkaisua, kun taas GPT-3.5 selvisi yksinkertaisemmalla tavalla. [28]

4 Suuret kielimallit frontend-kehityksessä

Tässä luvussa tarkastellaan edellisessä luvussa käsiteltyjen menetelmien käyttösovelluksia frontend-kehityksessä. Luvun tavoitteena on kattaa ymmärrys mallien mahdollisuuksista frontend-kehityksen, eli käyttöliittymien (engl. user interface, UI) ja käyttäjäkokemuksen (engl. user experience, UX), näkökulmasta. Luvussa vastataan samalla ensimmäiseen tutkimuskysymykseen: *Miten suuria kielimalleja voidaan hyödyntää frontend-kehityksen apuvälineinä?*

4.1 Generointi ja UI/UX-suunnittelu

Yleisimmät ohjelmointikielet frontend-kehityksessä tällä hetkellä ovat HTML, JavaScript ja CSS. Kielimallit, kuten ChatGPT ja Copilot, tukevat tehokkaasti näiden ohjelmointikielien generointia. Guo ym. [29] tarkastelivat ChatGPT:n kykyä generoida käyttäjätunnistautumiseen perustuva web-sovellus. Frontend-osiossa mallille annettiin tehtäväksi luoda kirjautumissivu, jonka tuli sisältää syötekentät käyttäjänimelle ja salasanalle sekä kirjautumispainike. Tuloksena ChatGPT tuotti rakenteellisesti selkeän HTML-sivun, josta ilmeni asianmukaiset elementit sekä frontend-puoli oli yhdistettävissä palvelinpuolen ohjelmalogiikkaan. Ohjelmakoodi osoittautui tutkimuksessa hyvin jäsennellyksi, selkeästi kommentoiduksi ja helposti muokattavaksi, mikä tukee mallien hyödyntämistä käyttöliittymäkehityksessä. Tutkimuksessa huo-

mattiin, että mallin suorituskyky ja kyvykkyys heikkenee, jos annetut ohjeet ovat liian epämääräisiä tai lyhyitä. Esimerkiksi kirjautumislomakkeen yhdistämisen palvelinpuolelle vaati erillisen tarkentavan kehotteen mallille.

Suuret kielimallit tarjoavat uusia mahdollisuuksia käyttöliittymien ja käyttäjäkokemuksen suunnittelussa. Mallien avulla voidaan automatisoida esimerkiksi käyttöliittymien tuottamista luonnoksista, vaihtoehtoisten asetteluratkaisujen luomista sekä käyttöliittymien visuaalisuuden ja käytettävyyden optimoimista.

Baulé ym. [30] havaitsivat, että nykyajan syväoppimiseen ja konvoluutioverkkoihin perustuvat generatiiviset mallit kykenevät muuntamaan käsin piirrettyjä luonnoksia suoraan HTML- ja CSS-muotoiseksi ohjelmakoodiksi. Näin nopeutetaan huomattavasti ohjelmistokehitystä, jolloin kehittäjä voi keskittyä käyttöliittymän ja vuorovaikutuksen suunnitteluun sen sijaan, että käyttäisi aikaa näiden manuaaliseen ohjelmointiin.

Jovanić ja Čarapina [31] havaitsivat, että mallit voivat auttaa suunnitteluprosessissa generoimalla valmiita komponenttiehdotuksia luonnollisesta kielestä. Mallit voivat myös ehdottaa erilaisia käyttöliittymäratkaisuja mahdollistaen vaihtoehtoisten käyttöliittymien vertailun kehityksen varhaisessa vaiheessa. Mallit kykenevät siis nopeuttamaan suunnitteluprosessia, vähentämään manuaalista työtä sekä mahdollistaa suuremman määrän vaihtoehtoisia käyttöliittymäratkaisuja esimerkiksi A/B-testaukselle. A/B-testauksessa asiakkaille näkyvästä sivusta luodaan kaksi eri versiota, jotka sitten jaetaan satunnaisesti näytettäväksi eri asiakkaille. Näin voidaan mitata asiakkaiden käyttäytymisen, kuten ostohalukkuuden, muutoksia sivulla eri käyttöliittymillä.

4.2 Testaus ja virheenetsintä

Ohjelmistotestauksen automatisointia suurilla kielimalleilla voidaan myös sovelluttaa frontend-kehitykseen. Perinteiset testausmenetelmät, kuten manuaalinen UI-testaus ja automaatiotestit, ovat usein työläitä ylläpitää nopeasti muuttuvissa sovellusympäristöissä. Suurten kielimallien avulla voidaan automatisoida testitapausten luominen, suoritus sekä validointi, jolloin sopeutuminen ohjelmistojen jatkuviin muutoksiin on joustavampaa.

Ale [32] esitteli frontend-testauksen automatisointiin luodun kehyksen. Kehys hyödynsi GAN-malleja (Generative adversial network), vahvistusoppimista ja suuria kielimalleja. Kehyksen avulla kyettiin generoimaan autonomisesti kattavia testitapauksia frontend-sovelluksille analysoimalla luonnollisella kielellä kirjoitettuja vaatimuksia ja käyttäjätarinoita. Näin mahdollistettiin testiskenaarioiden tuottaminen harvinaisemmilla käyttötilanteilla ja reunatapauksilla. Verrattuna perinteisiin testausmenetelmiin, tulokset osoittivat, että mallien avulla saavutettiin 20 % parempi virheiden tunnistustaso, 30 % lyhyempi suoritusaika testeille sekä 10 % laajempi testikattavuus.

Frontend-kehityksessä virheiden tunnistaminen ja korjaaminen eli debuggaus on keskeinen osa laadukkaan ja toimivan käyttöliittymän rakentamisessa. Le ym. [33] arvioivat ChatGPT:n ja Bardin kykyjä löytää ja korjata JavaScript-ohjelmointikielellä kirjoitetusta ohjelmakoodista haavoittuvuuksia. Tutkimuksessa käytettiin kolmea erilaista syötemuotoa: kontekstitonta, kontekstisensitiivistä ja kontekstirikasta kehotetta, jossa annettiin vaihtelevasti taustatietoa korjattavasta virheestä. Tulokset osoittivat, että molemmat mallit kykenivät korjaamaan suurimman osan virheistä: ChatGPT:llä 71,66 % onnistumisprosentti ja Bardilla 68,33 %. Tutkimuksessa havaittiin myös, että syötteen sisältämän kontekstin määrä vaikutti suoraan mallien onnistumiseen. Mitä enemmän mallille annettiin taustatietoa ja selityksiä virheestä, sitä todennäköisemmin malli onnistui korjaamaan virheen onnistuneesti. Tutki-

muksessa havaittiin myös, että yksinkertaisten virheiden, kuten indeksointi- tai validointivirheiden, korjaus oli tehokasta, mutta monimutkaisemmat ongelmat vaativat lisätietoja ja ohjelmoijan jälkitarkastusta. Malleilla voidaan siis tukea merkittävästi frontend-kehityksen debuggausta, vähentää manuaalista työtä, nopeuttaa kehitystä ja tukea kehittäjien tuottavuutta. Mallien suorituskyky riippuu kuitenkin annetun syötteen laadusta ja taustatiedon määrästä.

4.3 Kyvykkyys frontend-kehityksessä

Mallien kyvykkyyttä frontend-kehityksessä on tutkittu paljon viime aikoina, ja mallien kyvykkyys on edistynyt suuresti multimodaalisten mallien kehityksen myötä. Si ym. [34] toivat esille, kuinka nykyaikaset suuret kielimallit kykenevät muuttamaan visuaalisia käyttöliittymäluonnoksia suoraan toimivaksi HTML- ja CSS-koodiksi.

Tutkimuksen perusteella mallien kyvykkyyksissä ilmeni vahvuuksia sekä rajoituksia. Parhaat mallit, kuten GPT-4o, onnistuivat tuottamaan rakenteellisesti ja visuaalisesti hyvin samankaltaisia verkkosivuja annettujen mallikuvien pohjalta. Näissä tapauksissa mallien suoritustaso oli niin korkea, että arvioijat pitivät mallin tuottamaa sivua alkuperäistä parempana lähes 64 % tapauksissa. [34]

Haasteita ilmeni erityisesti monimutkaisissa tapauksissa. Frontend-sivustot, joissa oli suuret määrät HTML-tageja, syviä DOM-puurakenteita tai epätyypillisiä asetteluratkaisuja, aiheuttivat vaikeuksia myös edistyneille malleille. Lisäksi tutkimuksessa kävi ilmi, kuinka mallit saattoivat hallusinoida uusia elementtejä sekä asettaa alkuperäisiä elementtejä väärin tai jopa poistaa niitä. Tämä korostaa kehittäjän roolia laadunvarmistukselle. [34]

5 Pohdinta

Tässä luvussa pohditaan suurten kielimallien etuja, haasteita sekä kyvykkyyttä frontend-kehityksessä kolmannen sekä neljännen luvun pohjalta. Samalla pyritään vastaamaan tutkimuskysymyksiin: "Millaisia etuja tai haasteita suurten kielimallien käyttö tuo frontend-kehitykseen?"ja "Kuinka tarkkoja ja luotettavia suurten kielimallien tuottamat ohjelmointiratkaisut ovat?"

5.1 Suurten kielimallien edut

Suurten kielimallien hyödyntäminen frontend-kehityksessä tuo mukanaan etuja, jotka parantavat mahdollisesti sekä kehitystyön tehokkuutta, että lopputuloksen laatua. Ensimmäinen keskeinen etu on työn nopeuttaminen. Luvussa 4.1 todettiin, että mallit kykenevät tuottamaan valmista frontend-ohjelmakoodia, kuten HTML-, CSS- ja JavaScript-koodia, luonnollisen kielen kuvausten perusteella tai piirretystä luonnoksesta. Näin vähennetään manuaalista työn määrää rutiininomaisissa tehtävissä, jolloin kehittäjien ei tarvitse manuaalisesti määritellä tyylejä, dokumentoida ohjelmistoa sekä yksittäisiä funktioita tai ohjelmoida peruskomponentteja, kuten lomakkeita tai yksittäisiä painikkeita.

Toinen etu on luovuuden tukeminen. Luvussa 4.1 todettiin, että mallit voivat ehdottaa erilaisia lähestymistapoja käyttöliittymien toteuttamiseen. Näin voidaan laajentaa kehittäjien näkökulmia, helpottaa esimerkiksi A/B-testausta sekä optimoida käyttäjäkokemusta. Tämä mahdollistaa iteratiivisemman ja vapaamman kehityksen,

jossa kehittäjien ei tarvitse rakentaa manuaalisesti jokaista vaihtoehtoa alusta asti.

Suuret kielimallit voivat myös parantaa laadunvarmistusta ja virheiden korjaamista. Luvuissa 3.1.4 ja 4.2 todettiin, että mallit voivat tunnistaa syntaksi- ja logiikkavirheitä ohjelmakoodissa reaaliaikaisesti. Reaaliaikaiset korjausehdotukset virheille nopeuttavat niiden paikantamista sekä vähentävät virheiden päätymistä tuotantoon. Lisäksi mallien tuottamat automatisoidut testitapaukset voivat parantaa käyttöliittymien laatua sekä varmistaa, että lopputuote täyttää nykyaikaiset standardit.

Suurten kielimallien edut frontend-kehityksessä perustuvat tehokkuuden, laadun sekä luovuuden tukemiseen. Oikein hyödynnettynä mallit voivat toimia tehokkaasti kehittäjän tukena sekä parantaa koko kehitysprosessin laatua ja tuottavuutta.

5.2 Suurten kielimallien haasteet

Vaikka suuret kielimallit tarjoavat etuja frontend-kehityksessä, niiden hyödyntämisessä ilmenee myös useita haasteita. Keskeinen haaste on generoidun ohjelmakoodin luotettavuus. Mallien generoimassa ohjelmakoodissa voi esiintyä virheitä, kuten väärin toimivaa logiikkaa, saavutettavuusongelmia sekä huonoja ratkaisuja. Mallien tuottamissa vastauksissa voi myös ilmetä hallusinaatiota, jolloin uskottavan näköistä, mutta virheellistä ohjelmakoodia päätyy tuotantoon. Tämä saattaa lisätä virheiden syntyä ja siten myös ylläpitotöitä.

Toisena haasteena voi olla kontekstin hallinta. Lukujen 3.2 ja 4.3 pohjalta voidaan todeta, että mallit kykenevät tuottamaan hyviä ratkaisuja, kun niille annetaan riittävä konteksti, eli tieto ongelmasta. Ilman tarkkoja ja kontekstirikkaita syötteitä mallien vastaukset saattavat olla puutteellisia tai virheellisiä. Frontend-kehityksessä käyttöliittymien rakenne sekä toiminnallisuus palvelimen kanssa ovat monimutkaisia ja kontekstisidonnaisia tehtäviä. Tämä rajoittaa mallien käyttöä etenkin laajempien kokonaisuuksien hallinnassa.

Suurten kielimallien mukana tulee myös lukuisia muita haasteita, kuten ylläpidettävyys, laatu ja tietoturva. Vaikka malleilla kyetään tuottamaan nopeasti ohjelmakoodia, ratkaisut eivät välttämättä ole käytäntöjen mukaista esimerkiksi selkeyden tai skaalautuvuuden osalta. Tämä voi vaikeuttaa projektien pitkäaikaista kehittämistä, jos mallien tuottamaa ohjelmakoodia ei säännöllisesti arvioida. Mallien generoimassa ohjelmakoodissa voi myös esiintyä tietoturvariskejä etenkin käyttäjätietojen käsittelyssä. Käyttäjiä koskevien tietojen käsittelyssä ihmisen valvonta on siis välttämätöntä. Vaikka suuret kielimallit toimivat parhaimmillaan kehittäjän tukena, se ei poista itse kehittäjän vastuuta lopputuotteen laadun sekä turvallisuuden takaamisesta.

5.3 Suurten kielimallien kyvykkyys

Mallien kyvykkyys frontend-kehityksessä vaikuttaa tällä hetkellä lupaavalta erityisesti yksinkertaisten tehtävien osalta. Luvuissa 3.2 ja 4.3 käsitellyt tutkimukset osoittavat, että suuret kielimallit kykenevät tuottamaan toimivaa ohjelmakoodia useissa tehtävissä eri ohjelmointikielillä, kun malleille syötetään selkeä tehtävänanto ja riittävä konteksti. Mallit, kuten GPT-4o, kykenevät tuottamaan nopeasti rakenteellisesti toimivaa sekä visuaalisesti laadukasta ohjelmakoodia, mikä voi nopeuttaa merkittävästi käyttöliittymien luontia sekä kehityksen muita vaiheita.

Tutkimukset kuitenkin osoittavat, että mallien kyvykkyys ei ole vielä riittävä laajojen ja monimutkaisten frontend-projektien itsenäiseen toteutukseen. Suuret ohjelmistokokonaisuudet asettavat edelleen haasteita, kuten käytettävyyttä ja hallusinaatioita, joissa kehittäjän asiantuntemus sekä tarkka laadunvalvonta ovat välttämättömiä. Nämä korostavat kehittäjän merkitystä kehitysvaiheissa ja asettavat rajoja tekoälyn generoiman koodin luotettavuuteen. Vaikka suuret kielimallit toimivat tehokkaana apuvälineenä, lopullinen vastuu tuotteen laadusta kuuluu kehittäjille.

6 Yhteenveto

Tämän tutkielman tavoitteena oli selvittää, miten suuria kielimalleja voidaan hyödyntää frontend-kehityksessä, millaisia etuja ja haasteita sen käyttö tuo mukanaan sekä arvioida mallien tuottamien ohjelmointiratkaisujen tarkkuutta ja luotettavuutta. Tutkimusmenetelmänä käytettiin kirjallisuuskatsausta, jossa käytettiin ajankohtaisia tieteellisiä julkaisuja sekä konferenssiartikkeleita.

Tutkielmassa esiteltiin aluksi tekoälyn keskeiset menetelmät, suuret kielimallit sekä niiden hyödyntämistä ohjelmoinnissa yleisesti. Tämän jälkeen tarkasteltiin mallien sovelluskohteita frontend-kehityksessä. Sovelluskohteina tarkasteltiin ohjelmakoodin generointia, UI/UX-suunnittelua, ohjelmistotestausta sekä virheiden käsittelyä. Tutkielmassa havaittiin, että suuret kielimallit voivat nopeuttaa kehitysprosessia, tukea luovaa työskentelyä sekä parantaa laadunvarmistusta.

TK1 Generatiivisia tekoälymalleja voidaan hyödyntää frontend-kehityksessä esimerkiksi koodin generoinnissa, virheiden etsimisessä, dokumentaation tuottamisessa sekä tehtävänjaossa. Ne voivat toimia ohjelmoijan tukena kehitystyön eri vaiheissa.

TK2 Etuja ovat esimerkiksi ajansäästö, tehokkuuden kasvu, oppimismahdollisuudet ja inspiraation tarjoaminen. Haasteita ovat mahdollisesti koodin virheellisyys, hallitsematon sisältö, tietoturvariskit ja riippuvuus mallista. Myös eettiset ja oikeudelliset näkökohdat voivat muodostua haasteiksi.

TK3 Generatiivisten mallien ohjelmointiratkaisut voivat olla tarkkoja ja käyttökel-

poisia erityisesti yksinkertaisissa ja toistuvissa tehtävissä. Monimutkaisemmissa frontend-kokonaisuuksissa mallien tuottamissa ratkaisuissa voi esiintyä hallusinaatiota, rakennepuutteita tai virheellistä logiikkaa. Tästä syystä tuotetut ratkaisut vaativat aina kriittistä arviointia ja testausta ennen käyttöönottoa.

Yhteenvetona voidaan todeta, että suuret kielimallit tarjoavat frontend-kehitykselle merkittäviä mahdollisuuksia, mutta mallien hyödyntäminen ja niiden tuottamat tulokset vaativat kriittistä arviointia ja kehittäjän aktiivista osallistumista. Suurilla kielimalleilla voidaan saavuttaa tehokkaampaa ja laadukkaampaa kehitysprosessia, mutta vastuu laadusta ja turvallisuudesta pysyy kehittäjällä.

Lisäksi on tärkeää huomioida, että generatiiviset mallit ja niitä hyödyntävät työkalut kehittyvät erittäin nopeasti. Tämä kehitys tuo mukanaan jatkuvasti uusia mahdollisuuksia, mutta samalla myös uusia rajoitteita, kuten hallitsemattomia tai virheellisiä vastauksia, jotka voivat vaikuttaa kehitystyön laatuun. Mallien tehokas hyödyntäminen vaatii käyttäjältä edelleen kriittistä ajattelua ja teknistä osaamista.

Jatkotutkimuksena voitaisiin syventyä GenAI:n sovelluksiin, jossa ohjelmakoodia kyetään generoimaan kuvista tai käsin tehdyistä käyttöliittymäluonnoksista. Generoiminen suoraan kuvista tai piirroksista vaatii tarkkaa tutkimusta siitä, mitä vaikutuksia sillä mahdollisesti on kehittäjien sekä ohjelmistokehityksen rooliin tulevaisuudessa.

Lähdeluettelo

[1] R. Gozalo-Brizuela ja E. C. Garrido-Merchan, "ChatGPT is not all you need. A State of the Art Review of large Generative AI models", *arXiv preprint arXiv:2301.04655*, 2023.

[2] OpenAI, *ChatGPT: Optimizing Language Models for Dialogue*, `https://openai.com/blog/chatgpt`, 2022.

[3] GitHub, *GitHub Copilot: Your AI pair programmer*, `https://github.com/features/copilot`, 2021.

[4] Q. Ling, "Machine learning algorithms review", *Applied and computational engineering*, vol. 4, nro 1, s. 91–98, 2023.

[5] D. Jurafsky ja J. H. Martin, *Speech and Language Processing*, 3rd. Unpublished draft, 2024, Draft of January 12, 2025. url: `https://web.stanford.edu/~jurafsky/slp3/`.

[6] S. Pais, J. Cordeiro ja M. L. Jamil, "NLP-based platform as a service: a brief review", *Journal of Big Data*, vol. 9, nro 1, s. 54, 2022.

[7] D. Dodić ja D. Regodić, "Tokenization and Memory Optimization for Reducing GPU Load in NLP Deep Learning Models", *Tehnički vjesnik*, vol. 31, nro 6, s. 1995–2002, 2024.

[8] T. Mikolov, K. Chen, G. Corrado ja J. Dean, "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, 2013.

[9] J. Pennington, R. Socher ja C. D. Manning, ”Glove: Global vectors for word representation”, teoksessa *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, s. 1532–1543.

[10] K. R. Bokka, S. Hora, T. Jain ja M. Wambugu, *Deep Learning for Natural Language Processing: Solve your natural language processing problems with smart deep neural networks*. Packt Publishing Ltd, 2019, s. 75–118.

[11] P. Wulff, M. Kubsch ja C. Krist, ”Natural Language Processing and Large Language Models”, teoksessa *Applying Machine Learning in Science Education Research: When, How, and Why?*, P. Wulff, M. Kubsch ja C. Krist, toim. Cham: Springer Nature Switzerland, 2025, s. 117–142, ISBN: 978-3-031-74227-9. DOI: 10.1007/978-3-031-74227-9_7. url: https://doi.org/10.1007/978-3-031-74227-9_7.

[12] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema et al., ”A review on large language models: Architectures, applications, taxonomies, open issues and challenges”, *IEEE access*, vol. 12, s. 26 839–26 874, 2024.

[13] J. Devlin, M.-W. Chang, K. Lee ja K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL]. url: https://arxiv.org/abs/1810.04805.

[14] B. Min, H. Ross, E. Sulem et al., ”Recent advances in natural language processing via large pre-trained language models: A survey”, *ACM Computing Surveys*, vol. 56, nro 2, s. 1–40, 2023.

[15] J. Mahapatra ja U. Garain, ”Impact of model size on fine-tuned llm performance in data-to-text generation: A state-of-the-art investigation”, *arXiv preprint arXiv:2407.14088*, 2024.

[16] Z. Kasner ja O. Dušek, ”Beyond traditional benchmarks: Analyzing behaviors of open LLMs on data-to-text generation”, *arXiv preprint arXiv:2401.10186*, 2024.

[17] D. Zhang, Y. Yu, J. Dong et al., *MM-LLMs: Recent Advances in MultiModal Large Language Models*, 2024. arXiv: 2401.13601 [cs.CL]. url: https://arxiv.org/abs/2401.13601.

[18] S. Wu, Z. Peng, X. Du et al., ”A Comparative Study on Reasoning Patterns of OpenAI’s o1 Model”, *arXiv preprint arXiv:2410.13639*, 2024.

[19] L. Wang, C. Ma, X. Feng et al., ”A survey on large language model based autonomous agents”, *Frontiers of Computer Science*, vol. 18, nro 6, s. 186 345, 2024.

[20] Z. Ji, N. Lee, R. Frieske et al., ”Survey of hallucination in natural language generation”, *ACM computing surveys*, vol. 55, nro 12, s. 1–38, 2023.

[21] R. Song, Y. Li, L. Shi, F. Giunchiglia ja H. Xu, ”Shortcut Learning in In-Context Learning: A Survey”, *arXiv preprint arXiv:2411.02018*, 2024.

[22] R. Geirhos, J.-H. Jacobsen, C. Michaelis et al., ”Shortcut learning in deep neural networks”, *Nature Machine Intelligence*, vol. 2, nro 11, s. 665–673, 2020.

[23] C. Ebert ja P. Louridas, ”Generative AI for software practitioners”, *IEEE Software*, vol. 40, nro 4, s. 30–38, 2023.

[24] S. Greengard, ”AI rewrites coding”, *Communications of the ACM*, vol. 66, nro 4, s. 12–14, 2023.

[25] S. Dandotiya, ”Generative AI for software testing: Harnessing large language models for automated and intelligent quality assurance”, *International Journal of Science and Research Archive*, vol. 14, nro 1, s. 1931–1935, 2025.

[26] S. S. Dvivedi, V. Vijay, S. L. R. Pujari, S. Lodh ja D. Kumar, "A comparative analysis of large language models for code documentation generation", teoksessa *Proceedings of the 1st ACM International Conference on AI-Powered Software*, 2024, s. 65–73.

[27] F. A. Sakib, S. H. Khan ja A. R. Karim, "Extending the frontier of chatgpt: Code generation and debugging", teoksessa *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*, IEEE, 2024, s. 1–6.

[28] A. Agarwal, A. Chan, S. Chandel et al., "Copilot evaluation harness: Evaluating llm-guided software programming", *arXiv preprint arXiv:2402.14261*, 2024.

[29] M. Guo, "Java Web Programming with ChatGPT", teoksessa *2024 5th International Conference on Mechatronics Technology and Intelligent Manufacturing (ICMTIM)*, IEEE, 2024, s. 834–838.

[30] D. de Souza Baulé, C. G. von Wangenheim, A. von Wangenheim ja J. C. Hauck, "Recent Progress in Automated Code Generation from GUI Images Using Machine Learning Techniques.", *J. Univers. Comput. Sci.*, vol. 26, nro 9, s. 1095–1127, 2020.

[31] M. Jovanić ja M. Čarapina, "Application of Artificial Intelligence in the Creation of Web Content", teoksessa *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, IEEE, 2024, s. 2063–2068.

[32] N. K. Ale, "A Generative AI Framework for Enhancing Software Test Automation: Design, Implementation, and Validation",

[33] T. K. Le, S. Alimadadi ja S. Y. Ko, "A study of vulnerability repair in javascript programs with large language models", teoksessa *Companion Proceedings of the ACM Web Conference 2024*, 2024, s. 666–669.

[34] C. Si, Y. Zhang, R. Li, Z. Yang, R. Liu ja D. Yang, ”Design2Code: Benchmarking Multimodal Code Generation for Automated Front-End Engineering”, *arXiv preprint arXiv:2403.03163*, 2024.