



Tekoälyavustajat full stack -kehityksen tukena

Mahdollisuudet ohjelmistotekniikan opiskelussa

Lauri Palmén

OPINNÄYTETYÖ
Toukokuu 2024

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

PALMÉN, LAURI:

Tekoälyavustajat full stack -kehityksen tukena
Mahdollisuudet ohjelmistotekniikan opiskelussa

Opinnäytetyö 54 sivua, joista liitteitä 0 sivua
Huhtikuu 2024

Opinnäytetyön tarkoituksena oli vertailla neljän ajankohtaisen, suuriin kielimalleihin perustuvan tekoälyavustajan hyödyntämistä full stack -ohjelmointikehityksessä tietotekniikan opiskelijan näkökulmasta. Opinnäytetyön tutkimuskysymys on: mikä tekoälyavustaja soveltuu parhaiten tietotekniikan opiskelijan tueksi full stack -ohjelmointikehityksessä. Tavoitteena oli tuottaa blogialusta hyödyntäen eri tekoälyavustajia ja analysoida avustajien ominaisuuksia. Opinnäytetyö tehtiin yhteistyössä Tampereen ammattikorkeakoulun kanssa.

Opinnäytetyössä käsiteltävät tekoälyavustajat olivat OpenAI:n ChatGPT, Microsoft Copilot, GitHub Copilot sekä Google Gemini. Jokaista tekoälyavustajaa käyttäen kehitettiin toisiaan vastaavat blogialustat. Sovellusten tekoprosessien aikana analysoitiin tekoälyavustajien kykyä tukea ohjelmistokehitystä. Tekoälyavustajia vertailtiin frontendin, backendin ja tietokannan kehityksessä. Vertailun näkökulmia olivat koodin generointi, testidatan generointi, kokonaisuuksien suunnittelu ja avustajien helppokäyttöisyys.

Toteutuksen pohjalta todettiin, että kaikki tekoälyavustajat soveltuvat koodin generointiin. Eroja löytyi vastausten pituudessa, laadussa, ajankohtaisuudessa sekä avustajien käytettävyydessä.

Tekoäly kehittyi jatkuvasti ja tulokset ovat aikaan sidottuja. Työ keskittyi vain neljään valittuun tekoälyavustajaan. Tulokset ovat osittain subjektiivisia, sillä ne pohjautuvat tekijän kokemuksiin ja havaintoihin. Lisäksi avustajien kaikkia erikoisominaisuuksia ei tutkittu, jotta vertailu olisi helpompaa.

Asiasanat: tekoäly, suuret kielimallit, full stack, ohjelmistokehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

PALMÉN LAURI:
Leveraging AI Assistants in Full Stack Development
Opportunities in Software Engineering Studies

Bachelor's thesis 54 pages, appendices 0 pages
April 2024

The purpose of this thesis was to compare four topical AI assistants in full stack development from the perspective of an ICT student. The central question was which AI assistant provided the best support for ICT students in full stack development. The study was carried out as a blog platform project utilising OpenAI ChatGPT, Microsoft Copilot, GitHub Copilot and Google Gemini. The thesis was made in collaboration with Tampere University of Applied Sciences.

The blog platform was developed several times, each time using a different AI assistant. The capabilities and features of the AI tools were analysed during the development process.

The findings indicated that all the chosen AI assistants were suitable for code generation. Differences were found in the length of the responses, the quality of the code, the relevance of the provided information, and the usability of the assistants.

The evolving nature of AI means that the results are time-bound. Additionally, the results are partly subjective, as they are based on the author's experiences and observations. To simplify the comparison, not all the special features of the assistants were explored. Further research is needed to determine the full capabilities of the assistants.

Key words: artificial intelligence, large language models, full stack

SISÄLLYS

1	JOHDANTO	6
2	TEKOÄLY	8
	2.1 Tekoälyn määritelmä	8
	2.2 Tekoälyn kehitys	9
3	SUURET KIELIMALLIT	12
	3.1 Mikä on suuri kielimalli?	12
	3.2 Toimintaperiaatteet	13
	3.3 Suorituskyvyn mittaaminen	14
4	TEKOÄLYAVUSTAJAT	16
	4.1 Käyttötarkoitukset ja käyttö	16
	4.2 ChatGPT	17
	4.3 Google Gemini	19
	4.4 Microsoft Copilot	22
	4.5 GitHub Copilot	23
5	TEKOÄLYAVUSTAJIEN HYÖDYNTÄMINEN KEHITYKSESSÄ	26
	5.1 Hyödyt	26
	5.2 Projektin alustus	26
	5.3 Tietokanta	28
	5.4 Backend	32
	5.5 Frontend	38
6	Yhteenveto	45
7	POHDINTA	48
	LÄHTEET	50

ERITYISSANASTO tai LYHENTEET JA TERMIT (valitse jompikumpi)

IDE	Integrated Development Environment, ohjelmointiympäristö
RLHF	Reinforcement Learning from Human Feedback, ihmisen palautteeseen perustuva vahvistusoppiminen

1 JOHDANTO

Opinnäytetyön tarkoituksena oli vertailla neljän ajankohtaisen suuriin kielimalleihin perustuvan tekoälyavustajan kykyä tukea full stack -sovelluskehitystä tuottamalla yksinkertainen sovellus jokaista avustajaa hyödyntämällä. Työhön valitut avustajat olivat OpenAI:n ChatGPT, Google Gemini, Microsoft Copilot ja GitHub Copilot. ChatGPT:stä käytettiin maksullista GPT-4 versiota ja Google Geminista maksullista Ultra 1.0 mallia. GitHub Copilotista käytössä oli opiskelijan versio. Microsoft Copilotista käytettiin Tampereen ammattikorkeakoulun tarjoamaa, ominaisuuksiltaan ilmaista versiota vastaavaa, versiota.

Tekoälyavustajien käyttö ohjelmistokehityksessä yleistyy nopeasti. Vuonna 2023 Stack Overflown kyselyyn vastanneista merkittävä osa teki töitä full stack kehittäjän nimikkeellä. Seitsemänkymmentä prosenttia kyselyn tekoälyosuuteen vastanneista oli joko käyttänyt tekoälyä hyödykseen tai suunnitteli käyttävänsä sitä tulevana vuonna. (Stack Overflow 2023.) On siis ajankohtaista ja tärkeää selvittää tekoälyavustajien käyttöä myös ohjelmointikehityksen opiskelussa.

Kehittäjä, joka ei hyödynnä tekoälyä, saattaa olla tulevaisuudessa epäedullisessa asemassa ohjelmistokehityksen alalla. GitHubin tutkimusten mukaan GitHub Copilotia hyödyntäneet kehittäjät suorittivat tehtävänsä jopa 55 % nopeammin, kuin kehittäjät, jotka eivät käyttäneet tekoälyavustajaa (Kalliamvakou 2022). Tekoälyn yleistyessä alaa opiskelevien on tärkeää oppia hyödyntämään tekoälyavustajia opiskelun tukena. Tämä opinnäytetyö arvioi neljää ajankohtaista tekoälyavustajaa full stack -kehityksen tukemisessa opiskelijan näkökulmasta. Tuloksien perusteella voidaan arvioida eri avustajien sopivuutta opiskelun tueksi ja suositella eritasoisille opiskelijoille sopivia avustajia.

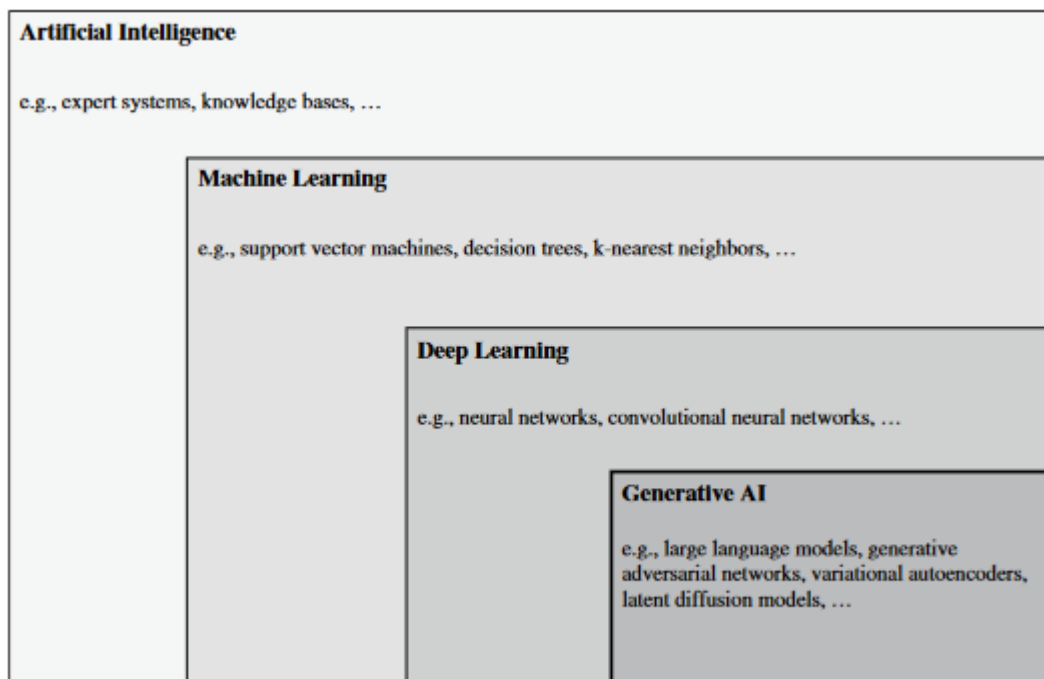
Suuria kielimallia hyödyntävien tekoälyavustajien suorituskykyä on testattu paljon useilla datasettitesteillä. Vaikka nämä testit antavat paljon korvaamatonta tietoa kielimallien suoriutumuksesta ja tehokkuudesta, niistä puuttuu usein konkreettinen analyysi esimerkiksi ohjelmoinnin työprosessiin liittyvistä seikoista. Tässä opin-

näytetyössä analysoidaan eri tekoälyavustajien sopivuutta tietotekniikan opiskelun näkökulmasta. Opinnäytetyö on tehty yhteistyössä Tampereen ammattikorkeakoulun kanssa.

2 TEKOÄLY

2.1 Tekoälyn määritelmä

Tekoäly (englanniksi Artificial Intelligence tai AI) on tieteen ala, jossa pyritään luomaan älykkäitä tietokoneohjelmia (McCarthy 2007, 2). Tekoälyllä tarkoitetaan myös teknologiaa, joka simuloi ihmisen älykkyyttä ja ongelmanratkaisutaitoja tietokoneissa. Tekoäly yhdistetään usein koneoppimiseen ja syväoppimiseen. (IBM Data and AI Team 2023.) Kuviossa 1 nähdään tekoälyn termien hierarkiaa ja termihin liittyviä konsepteja. Kuvion termit laajimmasta suppeimpaan ovat tekoäly, koneoppiminen, syväoppiminen ja generatiivinen tekoäly.



KUVIO 1. Tekoälyn termien hierarkiaa. (Banh & Strobel 2023, 2.)

Tekoäly voidaan jakaa kolmeen luokkaan: kapea tai heikko tekoäly (Artificial Narrow Intelligence, ANI), yleinen tekoäly (Artificial General Intelligence, AGI) ja supertekoäly (Artificial Super Intelligence, ASI). Kapea tekoäly pystyy tekemään tiettyjä tehtäviä, kuten tulkitsemaan puhuttua kieltä tai tunnistamaan kuvia. AGI pystyy ihmisen veroiseen ajatteluun ja ASI vielä kehittyneempään. Yleistä teko-

älyä ja supertekoälyä ei ole saavutettu vielä. (IBM Data and AI Team 2023.) Nykyiset suuria kielimalleja hyödyntävät generatiiviset tekoälyt ovat edistyneitä kaapeita tekoälyjä.

Generatiivisella tekoälyllä tarkoitetaan tekoälyä, joka voi luoda uusia tuotoksia opetusdatansa pohjalta. Tuotokset voivat olla tekstiä, kuvia tai muita datatyyppejä. Tekoälymallilta pyydetään tuotosta usein teksti promptilla eli kehotteella. (Martineau 2023.) Kehote on tekoälylle annettava, usein tekstimuotoinen, tehtävä (Microsoft Learn 2023). Kehotteet ovat usein tekstimuodossa, mutta ne voivat, mallista riippuen, olla myös koodinpätkiä, kuvia, ääntä tai videoita.

Tekoäly on kehittynyt lähivuosina paljon ja tekoälyn alalla on nähty paljon uusia termejä ja käyttötapauksia. Viimevuosina tekoäly on näkynyt kenties eniten suuria kielimalleja hyödyntävien generatiivisten chattibottien ja tekoälyavustajien muodossa.

2.2 Tekoälyn kehitys

Nykyaikaisen tekoälyn tutkimuksen juuret voidaan jäljittää 1940- ja 1950-luvulle. Turing ehdotti vuonna 1950 julkaisemassaan artikkelissa ”Computer Machinery and Intelligence” menetelmää koneen älykkyyden määrittelemiseksi. Menetelmä oli matkimispeli eli ”Imitation game”, jossa ihminen haastattelee sekä toista ihmistä että konetta tekstin välityksellä. Jos haastattelija ei pysty erottamaan koneen vastauksia ihmisen vastauksista, voidaan ajatella, että kone on älykäs. (Turing 1950, 433–434.) Peli tunnetaan nykyään keksijänsä mukaan Turingin testinä. Turingin testin luotettavuutta on kritisoitu. Kriitikoiden mukaan kone voi vaikuttaa älykkäältä testitilanteessa, vaikka sillä ei olisikaan todellisuudessa tarpeeksi tietoa ihmisen todelliseen imitoimiseen (McCarthy 2007, 4). Joidenkin lähteiden mukaan esimerkiksi nykyajan tekoälymalli ChatGPT 4 on läpäissyt Turingin testin tietyissä olosuhteissa (Mei, Xie, Yuan & Jackson 2024, 3).

Tekoälyn syntypaikkana pidetään Dartmouthin yliopistossa vuonna 1956 järjestettyä työpajaa. John McCarthy käytti ensimmäisen kerran termiä ”tekoäly” työ-

pajan nimessä: ”The Dartmouth Summer Research Project on Artificial Intelligence”. Alun perin termi oli tarkoitettu McCarthy ja Claude Shannonin kirjan ”Automata Studies” otsikoksi. (Kline 2011, 5–6). Työpajassa käsiteltiin älykkyyden ja oppimisen kuvailemista ja käsitteellistämistä tavalla, joka mahdollistaisi koneen simuloivan niitä. Keskeisiä tekoälyyn liittyviä aiheita olivat muun muassa koneiden automatisointi, puhutun kielen käyttäminen koneissa, sekä neuroverkkojen ja koneiden luovuus ja kyky kehittää itseään. (McCarthy, Minsky, Rochester & Shannon 2006, 12–14.) Dartmouthin työpaja oli merkittävä tapahtuma tekoälyn tutkimuksen kannalta. Kaikki Dartmouthin työpajassa käsitellyt aiheet ovat vielä nykypäivänä tekoälyn kannalta oleellisia.

Jospeh Weizenbaum kehitti ensimmäisen chattibotin, ELIZA:n, 1960-luvulla (Lo & Singh 2023, 201). Weizenbaum kuvaili ELIZA:a ohjelmaksi, joka mahdollistaa luonnollisen keskustelun tietokoneen kanssa. ELIZA analysoi syöttölauseita ja tunnisti avainsanoja, joiden pohjalta se rakensi vastauksen. (Weizenbaum 1966, 1). ELIZA ei pystynyt havaitsemaan kontekstia tekstistä eikä pitämään yllä pitkiä keskusteluja (Adamopoulou & Moussiades 2020, 2). Nykyaikaisiin järjestelmiin verrattuna ELIZA ja sen luonnollisen kielen prosessointi olivat yksinkertaisia, mutta ELIZA:lla oli varmasti vaikutusta nykyajan chattibottien kehitykselle (Lo & Sing 2023, 201–202).

1970 -ja 1980 luvulla yleistyivät asiantuntijajärjestelmät (expert systems). Asiantuntijajärjestelmät ovat tietokoneohjelmia, jotka pyrkivät emuloimaan alan asiantuntijan ongelmanratkaisutaitoja (Feinstein 1989, 182). Asiantuntijajärjestelmät pystyvät toimimaan omalla toimialallaan yllättävän hyvin. Yksi esimerkki hyvin erikoistuneesta asiantuntijajärjestelmästä on IBM:n Deep Blue shakkiohjelma. (Haenlein & Kaplan 2019, 8.) Vuonna 1997 IBM:n kehittämä tietokonejärjestelmä Deep Blue voitti sen aikaisen shakin maailmanmestarin Garry Kasparovin (Campbell, Hoane & Hsu 2002, 57, 59). Shakkia pidetään älyä vaativana pelinä ja siten pitkään ajateltiin, että kone, joka kykenisi pelaamaan shakkia osoittaisi älykkyyttä (Smith 2006, 10). Deep Blue:n voitto perustui kuitenkin enemmän suoritusnopeuteen kuin älykkyyteen. Asiantuntijajärjestelmät eivät kykene mukauttamaan tietotaitoaan oman alansa ulkopuolelle. (Haenlein & Kaplan 2019, 8.)

Vuonna 2015 Google DeepMindin AlphaGo voitti sen aikaisen maailmanmestarin lautapeli Go:ssa. Tämän uskottiin pitkään olevan mahdotonta, koska Go:ssa on huomattavan paljon mahdollisia liikkeitä. Esimerkiksi aloitusliikkeitä on Go:ssa 361 verrattuna shakin 20 liikkeeseen. AlphaGo pystyi tähän, koska se hyödynsi syväoppimista ja neuroverkkoja. (Haenlein & Kaplan 2019, 8.) AlphaGo koulutettiin osittain ammattilaisten pelien pohjalta ja osittain pelaamalla itseään vastaan. (Chen 2016, 4–6) AlphaGo osoitti tietokoneiden laskentatehon kasvua sekä neuroverkkojen ja syväoppimisen tehokkuutta.

Vuonna 2017 Google esitteli tutkimuspaperissa ”Attention is All You Need” Transformer-mallin, joka mahdollisti tiedon rinnakkaiskäsitellyn neuroverkoissa ja loi perustan tuleville tekoälyn kehityksille (IBM n.dc). Transformer-arkkitehtuurin paikallisen koodauksen (positional encoding) ja huomiomekanismin (self-attention) hyödyntäminen (Vaswani ym. 2017, 2, 6) tehostivat kielimallien luonnollisen kielien kontekstin ja sanojen välisten suhteiden ymmärtämisen kouluttamista (IBM n.dc).

Transformer-mallin pohjalta OpenAI toi vuonna 2020 markkinoille GPT-3-kielimallin. Se sisälsi edeltäviä kielimalleja huomattavasti enemmän parametreja (175 miljardia) (Brown ym. 2020, 1) ja pystyi generoimaan ihmisenkaltaista tekstiä, suorittamaan käännöksiä ja vastaamaan kysymyksiin vähäisillä syötteillä (Floridi & Chiriatti 2020, 684, 690). GPT-3 kielimallin erotti asiantuntijajärjestelmistä sen kyky suoriutua tehtävistä ilman erityistä koulutusta niihin (Brown ym. 2020, 1, 40–41). GPT-3:n monipuolinen kyky ymmärtää ja käyttää luonnollista kieltä asetti uuden standardin tekoälylle.

ChatGPT, joka on GPT-3.5-kielimallin version erikoistunut versio keskustelua varten, julkaistiin 2022 (OpenAI 2022). Se pystyi vastaamaan tekstikehoteisiin ihmisen kaltaisilla vastauksilla ja simuloimaan vuorovaikutusta entistä paremmin.

Kielimallien kasvu on ollut vauhdikasta 2020-luvulla. Nykypäivän markkinoilla on paljon GPT:n kanssa kilpailevia kielimalliperheitä, kuten esimerkiksi Google Gemini, Claude 3 ja LLaMA 2. Lisäksi on ilmestynyt kielimalleja hyödyntäviä tekoälyavustajia ja työkaluja kuten Microsoft Copilot ja GitHub Copilot.

3 SUURET KIELIMALLIT

3.1 Mikä on suuri kielimalli?

Suuri kielimalli eli LLM (Large Language Model) on kehittynyt neuroverkko, joka on koulutettu valtavalla määrällä dataa (Myers ym. 2024, 2). Kielimallit perustuvat koneoppimiseen, jossa käytetään Transformer-neuroverkkoarkkitehtuuria ja syväoppimista (Cloudflare n.d).

Suuret kielimallit ovat erityisen hyviä luonnollisen kielen prosessoinnissa ja soveltuvat siten hyvin moniin erilaisiin tehtäviin (Myers ym. 2024, 1). IBM (n.da) listaa suurten kielimallien tärkeiksi käyttökohteiksi muun muassa

- tekstin generoinnin
- tekstisisällön tiivistämisen
- tekoälyavustajat chattibottien muodossa
- koodin generoinnin
- tunteiden analysoinnin tekstin pohjalta
- kääntämisen kieleltä toiselle

Suurien kielimallien tuottamaan dataan ei voi aina luottaa. Malleilla on taipumus ”hallusinointiin”, missä kielimalli esittää vastauksen totuudenomaisena, mutta todellisuudessa faktat ovat puutteellisia (Myers ym. 2024, 17). Hallusinaatiot voivat johtua esimerkiksi ylioppimisesta tai opetusdatan vinoumista (biaksista), joiden takia malli havaitsee olemattomia yhteyksiä asioiden välillä. Hallusinaatioita voidaan vähentää muun muassa käyttämällä parempaa testidataa ja kehittämällä mallia jatkuvasti. (IBM. n.db). Monet tekoälyjärjestelmät perustuvat samoihin perusmalleihin (foundation model) ja saattavat siten periä niiden biakset (Myers ym. 2024, 4).

Suurien kielimallien hyödyntäminen on kasvanut lähiaikoina. Monet modernit tekoälyavustajat hyödyntävät suuria kielimalleja luonnollisen kielen prosessointiin ja vastausten tuottamiseen.

3.2 Toimintaperiaatteet

Suuret kielimallit perustuvat koneoppimiseen. Koneoppiminen on osa tekoälyä (IBM Data and AI Team 2023). Se on prosessi, jossa kehitetään algoritmeja, jotka voivat "oppia" tekemään tehtäviä sen sijaan, että koneelle kerrottaisiin suoraan, miten suorittaa tehtävä (Burzykowski, Rousseau, Geubbelmans & Valkenborg 2021, 732).

Koneoppiminen voidaan jakaa ainakin kolmeen kategoriaan: ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen (IBM Data and AI Team 2023). Ohjatussa oppimisessa mallille annetaan koulutusvaiheessa dataa, joka sisältää oikeat vastaukset. Kone oppii oikeiden vastausten pohjalta, miten ongelma ratkaistaan. Ohjaamattomassa oppimisessa opetusdatassa ei ole ratkaisuja mukana. Koneen pitää keksiä ratkaisut itse datassa toistuvien piirteiden pohjalta. (Louridas & Ebert, C. 2016, 110, 113.) Vahvistusoppimisessa kone oppii saamalla palautetta ympäristöstä palkintojen tai rangaistusten muodossa (IBM Data and AI Team 2023). Suurissa kielimalleissa käytetään esikoulutuksessa ohjaamatonta oppimista sekä joissain tapauksissa lisäksi myöhemmin ihmisen palautteeseen perustuvaa vahvistusoppimista (RLHF) (Myers ym. 2024, 9, 17).

Koneoppimisessa käytetään neuroverkkoja, jotka jäljittelevät aivojen toimintaa. Verkot koostuvat neuroneista ja kerroksista. Tyypillisesti neuroverkossa on syötekerros, useita piilotettuja kerroksia ja ulostulokerros. Neuroneilla on painoarvo ja kynnyсарvo, jotka vaikuttavat neuronin aktivoitumiseen. Neuronit ovat yhteydessä toisiinsa. Kun neuronin kynnyсарvo ylittyy syötteen takia, neuroni aktivoituu ja lähettää signaalin seuraavalle kerrokselle. (IBM Data and AI Team 2023.) Neuroverkon oppiminen tapahtuu, kun verkko oppii asettamaan oikeita painoarvoja yhteyksille, että saadaan haluttu lopputulos. Painoarvot muuttuvat, kun neuronit muuntavat verkon kokonaisaktivointia. (Schmidhuber 2015, 86, 88.)

Neuroverkkojen yhteydessä puhutaan usein syväoppimisesta ja syvistä neuroverkoista. Termit syväoppiminen ja syvä neuroverkko viittaavat neuroverkkoon, jossa on yli kolme neuroverkkotasoa (IBM Data and AI Team 2023). Joskin joi-

denkin alan asiantuntijoiden mukaan selkeästä rajasta ei ole sovittu (Schmidhuber 2015, 88). Suuret kielimallit, kuten GPT-kielimallit, pohjautuvat syväoppimiseen (Lo & Sing 2023, 208).

Google esitteli vuonna 2017 Transformer-mallin (muuntaja), jota käytetään suurissa kielimalleissa. Toisin kuin vanhemmat CNN ja RNN neuroverkot, Transformer-arkkitehtuurissa syötteet käsitellään rinnakkain, mikä tekee kouluttamisesta nopeaa. (IBM n.dc.)

Transformer-malleissa on kaksi tärkeää ominaisuutta: paikallinen koodaus (positional encoding) sekä huomiomekanismi (self-attention). Paikallisen koodauksen avulla malli antaa lauseen merkeille (token) uniikin numeron, jonka avulla malli voi ottaa huomioon merkkien järjestyksen. Huomiomekanismi painottaa merkkejä lauseessa niin, että malli voi oppia mitä sanoja käytetään yhdessä. (IBM n.dc.)

3.3 Suorituskyvyn mittaaminen

Suurien kielimallien nopean kehityksen takia on tärkeää pystyä mittaamaan mallien kyvykkyyttä. Kielimallien kykyjen arvioimista varten on luotu useita suorituskykyä mittaavia datasettejä.

Yksi yleisesti käytössä ollut testi on MMLU (Massive Multitask Language Understanding). MMLU on suorituskykytesti, joka sisältää 57 tehtävää. Tehtävät koostuvat muun muassa matematiikasta, historiasta ja tietotekniikasta. (Hendrycks ym. 2020, 1) Nykykielimallit suoriutuvat MMLU testistä melko hyvin. GPT-4:n on raportoitu saavuttavan 86,4 % pisteet (OpenAI Team 2023, 7) ja Googlen Gemini Ultran jopa 90 % pisteet (Gemini Team Google 2023, 6).

Suuret kielimallit suoriutuvat hyvin tekstin generoinnista ja ymmärtämisestä. Nykyajan suuret kielimallit on koulutettu ymmärtämään useaa eri datatyyppeä kuten tekstiä, kuvia ja ääntä. (Wu ym. 2023, 1.) Näitä multimodaalisia suuria kielimalleja ovat esimerkiksi GPT-4 (OpenAI Team 2023, 1) ja Gemini Ultra 1.0 (Gemini Team Google 2023, 1).

Yksi multimodaalisuutta mittaava testi on MMMU (Massive Multi-discipline Multimodal Understanding). MMMU on suorituskkyttesti, joka koostuu 11 500 korkeakoulutason kysymyksestä. Kysymykset sisältävät useita erilaisia kuvatyyppejä kuten taulukoita, diagrammeja ja nuotteja. Tässä testissä GPT-4V (V tarkoittaen with Vision) ja Gemini Ultra saavuttavat 56 % ja 59 % suorituksen. (Yue ym. 2023, 1, 9.) Tämä indikoi, että Gemini Ultra on hieman parempi multimodaalisilta kyvyiltään kuin GPT-4V.

Ohjelmointikykyjä varten on luotu omia testejä. HumanEval on suorituskkyttesti, joka mittaa kielimallien kykyä ratkoa koodiongelmia Pythonia käyttäen (Chen ym. 2021, 1). HumanEval testissä GPT-4:n on raportoitu saavuttavan 67,0 % (OpenAI Team 2023, 7) suorituksen ja Gemini Ultran 74,4 % (Gemini Team Google 2023, 8).

Datasetteihin perustuvat suorituskkyttestit mittaavat kielimallien kyvykkyyttä tehtävissä. Tämän takia niitä ei voida helposti käyttää useita teknologioita ja kielimalleja hyödyntävien tekoälyavustajien suoraan vertailuun. Esimerkiksi GitHub Copilot ja Microsoft Copilot perustuvat GPT-4 kielimallin lisäksi muihin teknologioihin. On mahdollista, että nämä tekoälyavustajat suoriutuvat todellisuudessa eriävästi GPT-4 kielimallin tuloksista.

4 TEKOÄLYAVUSTAJAT

4.1 Käyttötarkoitukset ja käyttö

Suurien kielimallien mukautuvuuden takia on olemassa paljon niitä hyödyntäviä tekoälytyökaluja. Näitä työkaluja ovat esimerkiksi erilaiset chattibottien kaltaiset avustajat kuten ChatGPT ja Google Gemini, sekä kuvan generoinnin välineet kuten DALLE-3 ja Stable Diffusion. Ohjelmistokehityksen kannalta oleellisimpia ovat tekstin generointiin keskittyvät tekoälyavustajat. Sen takia tässä osiossa esitellään ChatGPT, Google Gemini, Microsoft Copilot ja GitHub Copilot.

Tekoälyavustajille on full stack -ohjelmointikehityksessä useita käyttötarkoituksia. Stack overflown (2023) kyselyn tekoälyn hyödyntämisen osiossa 82,55 % (37 726 vastanneesta) vastasi käyttävänsä tekoälyä koodin kirjoittamiseen. Vastanneista 48,89 % sanoi käyttävänsä tekoälyä avun saamiseen ja vianetsintään, sekä 34,37 % koodin dokumentointiin. Vain 13,52 % kertoi käyttävänsä tekoälyä projektin suunnitteluun. (Stack overflow 2023.) Lisäksi tekoälyä voidaan käyttää testidatan tai testien kirjoittamiseen. Tekoäly voi myös ehdottaa parannuksia koodiin tai uudelleen kirjoittaa ominaisuuksia yksinkertaisemmaksi ja luettavammaksi eli refaktoroida koodia.

Avustajat toimivat pääsääntöisesti ohjeistamalla tai kysymällä kysymyksiä tekstikehotteiden avulla (kuva 1).

You

Refactor this to use arrow functions and for of loop: `function printArray(arr) { for (let i = 0; i < arr.length; i++) { console.log(i, arr[i]); } }`

```
const arr = [1, 2, 3, 4, 5];
printArray(arr);
```

ChatGPT

You can refactor the "printArray" function to use an arrow function for its definition and a "for...of" loop to iterate through the array elements. Here's how you can do it:

```
javascript
const printArray = arr => {
  for (const element of arr) {
    console.log(i, element);
  }
};

const arr = [1, 2, 3, 4, 5];
printArray(arr);
```

This version uses "const printArray = arr => {...}" to define the function using an arrow function syntax. The "for...of" loop iterates directly over the array values, making it more concise and easier to understand than using a traditional for loop with an index. Each element of the array is referred to by "element" in the loop, and is passed to "console.log".

Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

This version uses "const printArray = arr => {...}" to define the function using an arrow function syntax. The "for...of" loop iterates directly over the array values, making it more concise and easier to understand than using a traditional for loop with an index. Each element of the array is referred to by "element" in the loop, and is passed to "console.log".

You

sort the array before printing it

ChatGPT

To sort the array before printing its elements, you can simply call the "sort()" method on the array before looping through it. Here's the modified version of the function which includes sorting:

```
javascript
const printArray = arr => {
  arr.sort((a, b) => a - b); // Sorts the array in ascending order
  for (const element of arr) {
    console.log(i, element);
  }
};

const arr = [5, 3, 1, 4, 2]; // Example array which is not initially sorted
printArray(arr);
```

In this version: ↓

Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

KUVA 1. ChatGPT 4:n ja käyttäjän interaktio pyydetessä muokkaamaan olemassaolevaa koodia.

Keskustelu rakentuu tekoälyn ja ihmisen välisestä vuorovaikutuksesta. Tähän poikkeuksena on GitHub Copilot, jonka pääkäyttötarkoitus on koodiehdotukset koodin seassa. Tässä opinnäytetyössä GitHub Copilotia käytettiin pääasiassa chat-käyttöliittymän kautta vertailukelpoisuuden takia.

4.2 ChatGPT

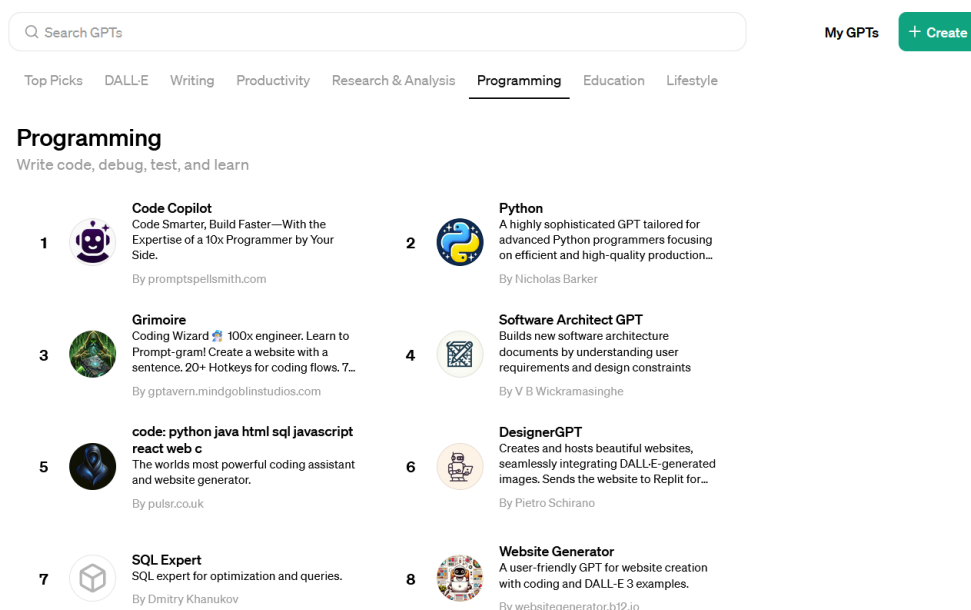
OpenAI julkaisi ChatGPT:n prototyypin marraskuussa 2022 (OpenAI 2022). ChatGPT on generatiivinen kielimalli, joka kykenee ihmisen veroiseen kirjoitetun tekstin tuottamiseen ja ylläpitämään keskustelua luonnollista kieltä käyttäen (Roumeliotis & Tselikas 2023, 1, 3). ChatGPT:sta on olemassa GPT-3.5 ja GPT-4 versio, jotka on muokattu OpenAI:n GPT-3.5 (OpenAI 2022) ja GPT-4 kielimallista.

ChatGPT-kielimallit perustuvat Transformer-neuroverkkoarkkitehtuuriin (Roumeliotis & Tselikas 2023, 3.). GPT tarkoittaa esikoulutettua generatiivista transformer -kielimallia (Generative Pre-trained Transformer) (Yenduri ym. 2023, 1). ChatGPT:n kyky luoda ja ymmärtää luonnollista kieltä perustuu massiiviseen tekstipohjaiseen koulutusdataan, joka syötettiin mallille esikoulutusvaiheessa. Koulutukseen käytettiin ohjaamatonta oppimista ja sen tarkoituksena oli opettaa

mallille luonnollisen kielen struktuuria sekä sanojen ja lauseiden välisiä yhteyksiä. (Roumeliotis & Tselikas 2023, 1–2.) Lisäksi koulutukseen käytettiin vahvistusoppimista (OpenAI 2022).

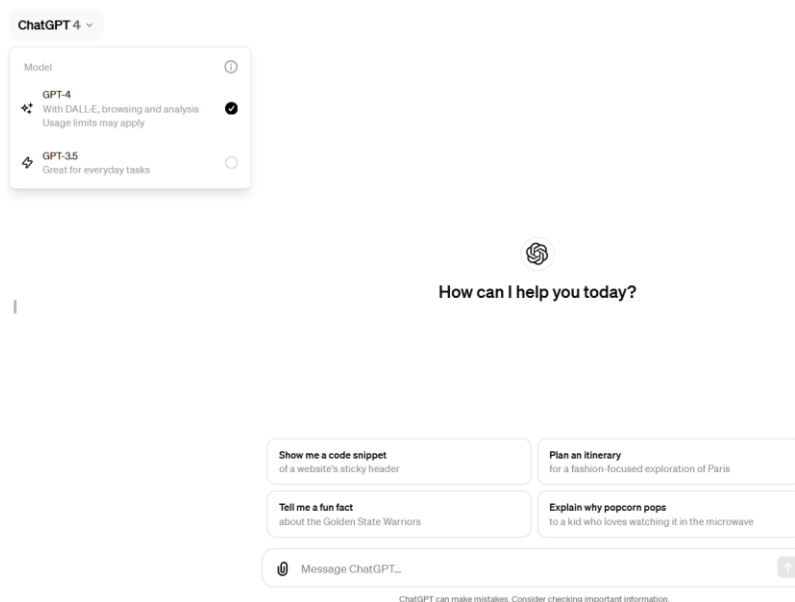
GPT-4 on OpenAI:n viimeisin kielimalli. Se on multimodaalinen ja pystyy siksi edellisen mallin tekstiminaisuuksien lisäksi analysoimaan kuvia (OpenAI Team 2023, 1). GPT-4 suoriutuu GPT-3.5:a paremmin muun muassa ihmiselle suunnitelluissa kokeissa ja luonnollisen kielen suorituskäytöissä. (OpenAI 2023, 1, 5–6). Tässä työssä käytettiin ChatGPT:n GPT-4 versiota.

GPT-4 versiolla voidaan hyödyntää myös yhteisön luomia kustomi GPT:tä. Näille malleille on annettu ohjeita ja ylimääräistä tietoa etukäteen (OpenAI 2023). Kustomi GPT:t voivat vastata tietyllä tavalla automaattisesti ja olla sen takia hyödyllisiä erikoistuneissa tehtävissä. ChatGPT:n hausta voit valita esimerkiksi SQL:n kirjoittamiseen valmistellun GPT:n (kuva 2).



KUVA 2. Kustomi GPT:n hakunäkymä.

ChatGPT:tä voi käyttää käyttäjäystävällisen verkkosivukäyttöliittymän kautta. Käyttöliittymässä voidaan valita GPT 3.5 version ja maksullisen GPT 4 version väliltä (kuva 3).



KUVA 3. ChatGPT:n käyttöliittymä. GPT-version valinta vasemmasta ylänurkasta. Kysymysten kirjoittamiseen ja liitteiden lisäämiseen kenttä keskellä alhaalla.

ChatGPT:n 3.5 versiolle voi syöttää vain tekstiä ja se vastaa myös vain tekstinä. GPT 4 versio voi analysoida myös kuvia sekä generoida niitä. Kuvan generointiin ChatGPT 4 käyttää OpenAI:n DALL-E 3 järjestelmää. ChatGPT generoi yhden vastauksen, mutta se voidaan myös uudelleen generoida tarvittaessa.

4.3 Google Gemini

Gemini on Google DeepMindin luoma tekoälymalliperhe. Gemini 1.0 versiosta on kolme mallia: Ultra, Pro ja Nano. Eri versiot soveltuvat suorituskyvyltään erilaisiin tarkoituksiin. Ultra on kykenevin, mutta vie eniten resursseja. Nano on tarkoitettu vähemmän muistia sisältäviin laitteistoihin. Pro on tasapainossa suorituskyvyn ja muistin käytön välillä (Gemini Team Google 2023, 1.) Lisäksi Pro versiosta on julkaistu joihinkin maihin versio 1.5, joka lupaa Ultra versiota parempaa suorituskyyä (Reid ym. 2024, 1, 3). Tässä työssä käytettiin Ultra 1.0 versiota.

Myös Gemini-mallit perustuvat Transformer-arkkitehtuuriin. Mallit on koulutettu tekstillä, kuvilla, audiolla ja videoilla. (Gemini Team Google 2023, 1, 3, 5.) Gemini on koulutettu multimodaaliseksi ja monikieliseksi (Gemini Team Google 2023, 5).

DeepMind väittää Gemini 1.0 Ultra version voittavan OpenAI:n GPT-4:n useissa suorituskykytesteissä. Huomattavasti esimerkiksi kuvan 4 ensimmäisellä rivillä näkyvässä moniosaamista testaavassa MMLU testissä. (Gemini Team Google 2023, 6; Pichai & Hassabis 2023.)

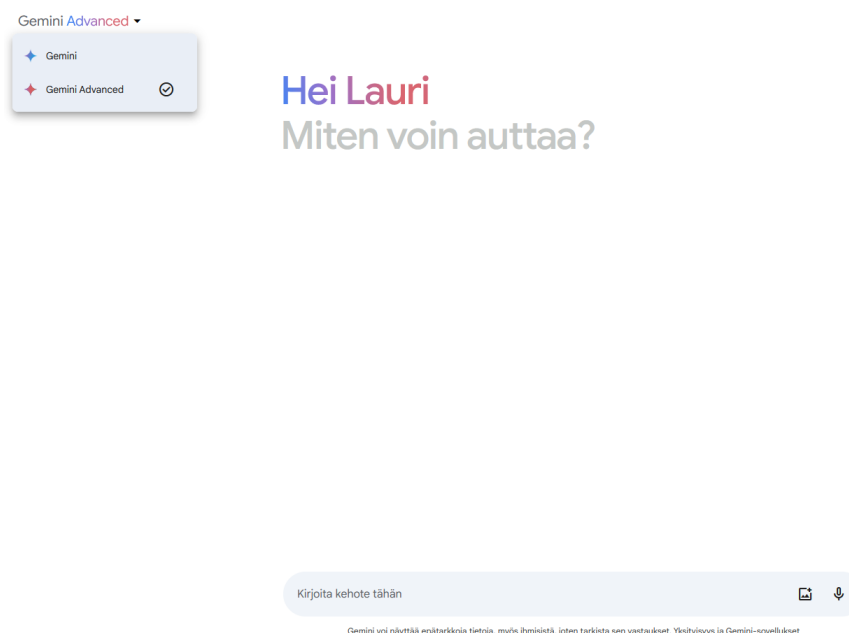
TEXT			Gemini Ultra	GPT-4
Capability	Benchmark Higher is better	Description		API numbers calculated where reported numbers were missing
General	MMLU	Representation of questions in 57 subjects (incl. STEM, humanities, and others)	90.0% CoT@32*	86.4% 5-shot** (reported)
Reasoning	Big-Bench Hard	Diverse set of challenging tasks requiring multi-step reasoning	83.6% 3-shot	83.1% 3-shot (API)
	DROP	Reading comprehension (F1 Score)	82.4 Variable shots	80.9 3-shot (reported)
	HellaSwag	Commonsense reasoning for everyday tasks	87.8% 10-shot*	95.3% 10-shot* (reported)
Math	GSM8K	Basic arithmetic manipulations (incl. Grade School math problems)	94.4% maj1@32	92.0% 5-shot CoT (reported)
	MATH	Challenging math problems (incl. algebra, geometry, pre-calculus, and others)	53.2% 4-shot	52.9% 4-shot (API)
Code	HumanEval	Python code generation	74.4% 0-shot (IT)*	67.0% 0-shot* (reported)
	Natural2Code	Python code generation. New held out dataset HumanEval-like, not leaked on the web	74.9% 0-shot	73.9% 0-shot (API)

* See the technical report for details on performance with other methodologies

** GPT-4 scores 87.29% with CoT@32 - see the technical report for full comparison

KUVA 4. Gemini Ultra vastaan GPT-4 valituissa testeissä. (Pichai & Hassabis 2023)

Geminita voi käyttää esimerkiksi Googlen verkkosivukäyttöliittymän kautta. Geminilta voi kysyä chattikäyttöliittymän kautta vastauksia tekstillä, äänellä tai kuvina. Geminin suurin kielimalli Ultra 1.0 on valittavissa käyttöliittymän vasemmasta ylänurkasta (kuva 5), jos Google käyttäjällä on aktiivinen Google One -tilaus.



KUVA 5. Geminin käyttöliittymä. Version valinta vasemmassa ylänurkassa.

Gemini-malli ymmärtää videoita (Gemini Team Google 2023, 15), mutta tätä ei tueta chattikäyttöliittymässä. Gemini vastaa kysymyksiin vain tekstinä. Gemini luo vastauksista kolme luonnosta ja ne voidaan generoida myös uusiksi.

Geminin vastauksia voi arvioida sisäänrakennetun ”Tarkista vastaus huolellisesti”-napin kautta. Nappia painaessa järjestelmä tarkistaa Geminin väitteitä Googlen hakua käyttämällä. Kuvassa 6 Geminin tarkistusnappi käyttöliittymässä vastauksen alla.



KUVA 6. Gemini-käyttöliittymän ”tarkista”-nappi.

Vastauksen tarkistuksen jälkeen käyttöliittymä korostaa oikean oloiset vastaukset vihreällä ja mahdollisesti väärät oranssilla, sekä näyttää alla lähteen (kuva 7). Haku ei aina merkitse kaikkia väitteitä.

1. Fact-Based Questions - General Knowledge

- **Question:** Who invented the telephone?
- **Answer:** The telephone is widely credited to Alexander Graham Bell. ^

Google Haku löysi samantyyppistä sisältöä, esimerkiksi tämän:

Who is credited with inventing the telephone? | Library of Congress

Alexander Graham Bell is credited with being the inventor of the telephone since his patent and demonstrations for an apparatus designed for "transmitting ...

 www.loc.gov



4. Current Events and Pop Culture

- **Question:** What was the top-grossing movie of 2023?
- **Answer:** The top-grossing movie of 2023 was Avatar: The Way of Water. !

Google Haku löysi toisenlaista sisältöä, esimerkiksi tämän:

The Ten Highest-Grossing Movies at the Global Box Office in 2023 - Boxoffice

The Super Mario Bros. is the second-highest-grossing film of 2023 on both the domestic and international charts, with the animated video game adaptation ...

 www.boxofficepro.com

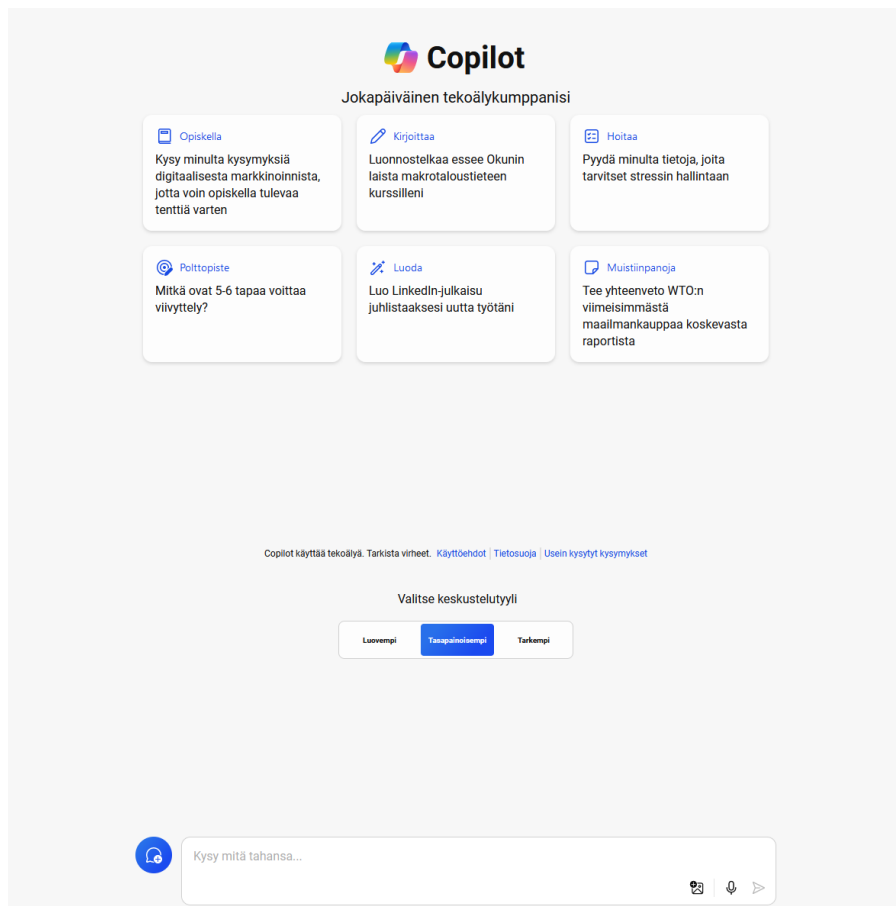


KUVA 7. Geminin Googlen avulla tarkistetut vastaukset. Ylhäällä oikeaksi merkitty vastaus vihreällä, ja alhaalla vääräksi merkattu vastaus oranssilla.

4.4 Microsoft Copilot

Microsoft Copilot on tekoälyavustaja, joka on integroitu Microsoftin tuotteisiin. Se yhdistää Microsoftin aikaisemmin lanseeraamia tuotteita kuten Bing Chat ja Microsoft 365 Copilot yhden termin alle. (Mehdi 2023b). Copilot perustuu yhdistelmään eri teknologioita. Se koostuu muun muassa OpenAI:n GPT-4-kielimallista sekä Microsoftin omista yksityisistä teknologioista. (Microsoft Learn 2024a.)

Copilotin erottaa muista avustajista sen integroituminen Microsoftin ekosysteemin sovelluksiin, kuten Wordiin, Exceeliin ja Outlookiin (Mehdi 2023b). Copilot on käytettävissä muun muassa Bing-hakukoneella, Edge-selaimella, Windows 11 työkaluissa, sekä Microsoft 365 sovelluksissa. (Mehdi 2023b). Selaimen chatti-käyttöliittymän kautta Microsoft Copilot on hyvin samanlainen kuin esimerkiksi ChatGPT ja Gemini (kuva 8). Sille voi antaa teksti- tai äänikehotteen, tai ladata kuvan. Copilot voi vastata tekstillä tai kuvilla.



KUVA 8. Microsoft Copilotin verkkokäyttöliittymä. Keskustelutyylin valinta chattikentän yläpuolella.

Copilotin käyttöliittymässä on kolme vaihtoehtoa keskustelutyyliille: Luovempi, Tasapainoisempi ja Tarkempi (kuva 8). Microsoftin mukaan Luovempi keskustelutyyli sopii muun muassa tarinoiden kirjoittamiseen ja datan luovemmalla tavalla esittämiseen. Tarkempi tyyli sopii lyhyempiin ja tarkempiin vastauksiin, kuten vuosilukujen ja laskutoimitusten kysymiseen. Tasapainoinen tyyli yhdistää edelliset tyylit ja antaa kattavasti tietoa melko lyhyessä muodossa. (Microsoft Bing 2023.)

4.5 GitHub Copilot

GitHub Copilot on tekoälyavustaja, joka on suunniteltu erityisesti koodin generointia varten. Se perustuu OpenAI:n GPT-4 malliin (Dohmke 2023). GitHub Copilot on koulutettu erityisesti koodaukseen liittyviä kysymyksiä varten (GitHub n.da).

GitHub Copilotin erottaa muista tekoälyavustajista sen syvä integroituminen ohjelmointiympäristöihin. Sitä käytetään pääosin ehdotusjärjestelmänä, joka ennustaa ja täydentää koodia sitä kirjoittaessa (kuva 9).

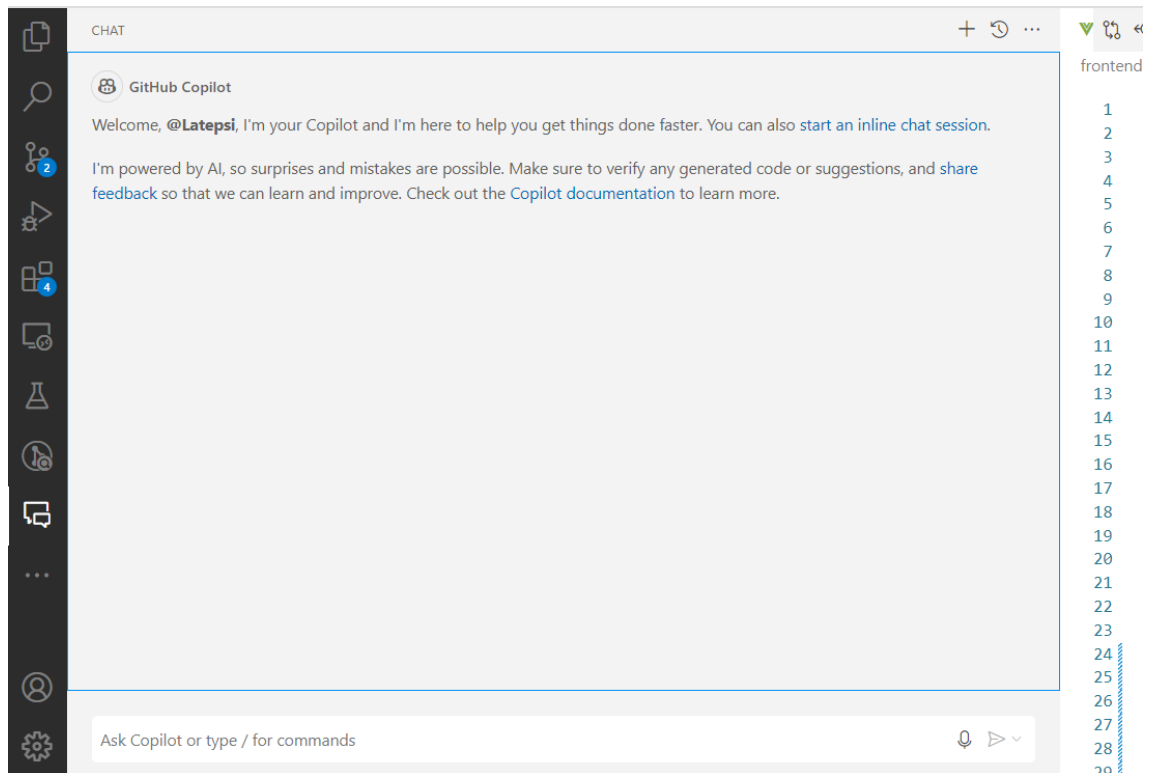
```

1
2 // a function that iterates over a list of numbers and returns the sum of all the numbers
3 function sum(numbers) {
    let total = 0;
    for (let i = 0; i < numbers.length; i++) {
        total += numbers[i];
    }
    return total;
}

```

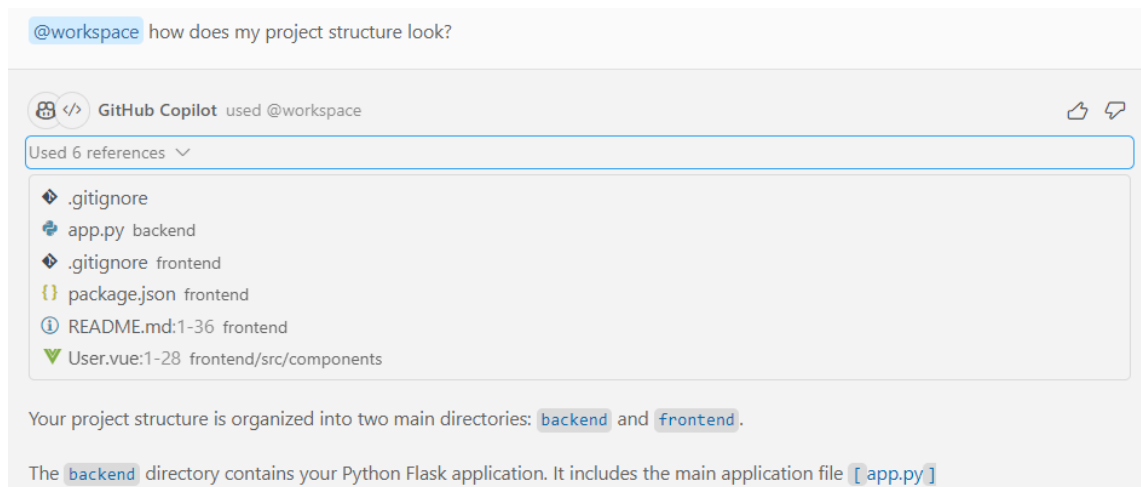
KUVA 9. GitHub Copilot ehdottaa funktiorakennetta koodin seassa kommentin ja edeltävän koodin perusteella.

GitHub Copilotissa on myös chattikäyttöliittymä, johon pääsee käsiksi esimerkiksi Visual Studio Code IDE:ssa lisäosan kautta (Dohmke 2023) (kuva 10). Chattikäyttöliittymä on hyvin samankaltainen kuin ChatGPT:n, Geminin ja Microsoft Copilotin verkkokäyttöliittymät.



KUVA 10. GitHub Copilotin chattikäyttöliittymä auki Visual Studio Code -IDE:n työtilassa.

Copilot pystyy katsomaan työtilassa olevia kooditiedostoja ja ehdottamaan siten tapauskohtaisia ratkaisuja koodin pohjalta. Visual Studio Code IDE:ssa komenolla `@workspace` (kuva 11) Copilot voi tutkia työhakemistoa ja valita kysymykseen olennaisia tiedostoja parempaa vastausta varten. Visual Studio Code IDE:hen liittyvissä asioissa voi käyttää `@vscode`-komentoa (GitHub Blog 2023.) Lisäksi käytössä on `@terminal`-komento, jolla Copilot voi avustaa terminaaliin liittyvissä asioissa.



KUVA 11. GitHub Copilotin työtilasta löytämät tiedostot vastausta varten `@workspace`-komentoa käytettäessä.

GitHub Copilotista on olemassa Enterprise-versio, joka voi tutustua yrityksen GitHub koodikantoihin ja antaa siten projekti- tai yrityskohtaisia ratkaisuja (GitHub n.db). Enterprise versiossa chattikäyttöliittymää voi käyttää myös GitHubin verkkosivuilla (GitHub n.da).

5 TEKÖÄLYAVUSTAJIEN HYÖDYNTÄMINEN KEHITYKSESSÄ

5.1 Hyödyt

Tekoälyavustajia voidaan hyödyntää useissa tehtävissä, mukaan lukien full stack -ohjelmistokehityksessä. Suurten kielimallien kyky käsitellä luonnollista kieltä mahdollistaa niiden soveltamisen myös koodin generointiin, joka perustuu oman kaltaiseensa ohjelmointikieleen eli syntaksiin. Osa kielimalleista, kuten GitHub Copilotin kielimalli, on myös koulutettu nimenomaan koodin generointiin.

Kielimalleja voidaan hyödyntää full stack -kehityksessä usealla tavalla. Tekoäly voi auttaa esimerkiksi suunnittelemaan projektin rakenteen ja vaiheet. Generatiivisia kykyjä voidaan käyttää myös koodin tuottamiseen, selittämiseen, refaktointiin ja dokumentointiin. Tekoälystä voi olla hyötyä myös vianetsinnässä.

Blogialustan toteutuksessa tekoälyavustajia arvioitiin tietokannan, backendin ja frontendin rakentamisessa. Lisäksi tekoälyä käytettiin projektin suunnitteluun ja arvioitiin tekoälyavustajien hyödyntämisen yleistä käyttökokemusta.

5.2 Projektin alustus

Projektin alustus on kriittinen vaihe, jossa pitää valita oikea suunta ja rakenne. Opinnäytetyössä toteutettiin Blogialusta, jossa on mahdollista listata käyttäjiä ja heidän blogiartikkeleitaan. Sovellukset kehitettiin rinnakkain, ja jokaisen sovelluksen kehittämisessä hyödynnettiin yhtä tekoälyavustajaa. Projektin aloittamiseksi tekoälyavustajille annettiin seuraava kehoite:

```
You are a senior full stack developer with accurate information about programming and best practices. We are building a full stack web application consisting of Vue frontend, Flask backend and PostgreSQL database. The web application will be a "Blog platform" hosted locally. I am developing
```

this using WSL and VSCode. Your task is to help me develop this app. What should I do first?

Kehotteessa annettiin tekoälylle selkeä rooli ja taustatietoa teknologiasta ja aiheesta. Lopuksi annettiin tehtävä. Kielenä käytettiin englantia, koska joidenkin kielimallien, kuten Google Gemini Ultran, sivuilla mainittiin tuki vain englannin kielelle.

Kaikki kielimallit vastasivat yleiskatsauksella projektin vaiheista, mutta yksityiskohdissa oli lieviä eroja. ChatGPT ja Gemini tarjosivat yksityiskohtaisempia listoja vaiheista ja ehdottivat teknologioita tukevia VSCode-lisäosia. Microsoft Copilot keskittyi yleisiin vaiheisiin ja mainitsi Node ja npm vaatimukset, mitä muut tekoälyavustajat eivät tehneet. GitHub Copilot kertoi yleispiirteisesti projektista ilman mainintaa lisäosista.

ChatGPT ja Gemini ehdottivat mahdollista kansiorakennetta. GitHub Copilot ohjeisti kansiorakenteen rakennuksen komentorivi komennoilla, kuten nähdään kuvasta 12. Microsoft Copilot tarjosi kansiorakenteen vasta erikseen pyydettyäessä.

```
Here's a basic outline of the commands you might run to set up your projects:
```

```
# Create project directories
mkdir blog-platform
cd blog-platform
mkdir frontend backend

# Initialize Vue.js project
cd frontend
vue create .

# Initialize Flask project
cd ../backend
python3 -m venv env
source env/bin/activate
pip install flask

# Initialize Git repository
cd ..
git init
```

Remember to replace `blog-platform` with the name of your project. After running these commands, you should have a basic project structure set up and be ready to start developing your application.

KUVA 12. GitHub Copilotin komentorivi ohjeet projektin alustamiseen.

Yksikään tekoäly ei kertonut täydellisesti, miten Node, npm tai Pythonin pip pitäisi asentaa. Microsoft Copilot mainitsi vaatimukset ohimennen, mutta kaikkien oletus oli, että nämä vaatimukset on asennettu.

5.3 Tietokanta

Tietokannan pystyttämiseen hyödynnettiin PostgreSQL, joka on olio-relaatiotietokanta, missä käytetään SQL-kieltä (PostgreSQL n.d). Se valittiin, koska PostgreSQL on suosiota kasvattanut SQL-tietokanta. Tietokannan toteutuksessa haettiin testata tekoälyn kykyä avustaa tietokannan pystyttämisessä sekä SQL-kyselyiden ja testidatan generoinnissa.

Tietokannan asennus aloitettiin kysymällä tekoälyiltä ohjeita PostgreSQL palvelimen pystyttämiseen WSL:ää käyttäen kehoitteella

```
How do I setup a PostgreSQL server, I'm using WSL
```

Vastaukset sisälsivät ohjeet Linux-pakettien päivittämiseen, PostgreSQL-pakettien asentamiseen, palvelimen käynnistämiseen ja oletustietokantaan yhdistämiseen. Kuvassa 13 Microsoft Copilotin vastaus. Muiden vastaukset olivat samankaltaisia.

Sinä
How do I setup a PostgreSQL server, I'm using WSL

Copilot
Certainly! Setting up a PostgreSQL server in Windows Subsystem for Linux (WSL) is a great choice. Here are the steps to get you started:

1. Install WSL2:

- Open your WSL terminal.
- Update your Ubuntu packages:

```
sudo apt update
```

- Install PostgreSQL and the `-contrib` package (which includes helpful utilities):

```
sudo apt install postgresql postgresql-contrib
```


- Confirm the installation and check the version:

```
psql --version
```

2. Configure PostgreSQL:

- After installation, PostgreSQL should be running as a service. You can start, stop, or restart it using:

```
sudo service postgresql start
sudo service postgresql stop
sudo service postgresql restart
```



KUVA 13. Osa Microsoft Copilotin ohjeistuksesta PostgreSQL-palvelimen pystyttämiseksi käyttäen WSL.

Microsoft Copilot ja GitHub Copilot vaativat erillisen pyynnön, että ne antoivat ohjeet tietokantakäyttäjän luomiseen. ChatGPT ja Gemini sisällyttivät osuuden alkuperäiseen kysymykseen. Ohjeiden perusteella saatiin luotua toimiva tietokanta.

Seuraavaksi keskityttiin `Users`- ja `Articles`-taulujen luomiseen. Tekoälyille piti tarkoittaa, että käytössä on PostgreSQL, muuten esimerkiksi Gemini saattoi käyttää queryssa MySQL:än `auto_increment`-avainsanaa. ChatGPT ja Gemini selvisivät taulujen generoinnista parhaiten. Copilotit unohtivat yhdistää `Articles`-taulun `writer`-kentän `Users`-taulun `id`-kenttään, mutta korjasivat ongelman pyydettyäessä. Kuvassa 14 nähdään ChatGPT:n onnistunut vastaus, jossa artikkeleiden `writer`-kenttä on yhdistetty `Users`-tauluun viiteavaimella.

You

Can you generate a query for a table called Articles. An article has a writer, a title, a text body, and a publication date.

ChatGPT

Certainly! Here's a SQL query to create a table named `Articles` that includes the fields you specified. This query assumes that the `writer` field will reference the `id` of a user in the `Users` table, establishing a foreign key relationship between `Articles` and `Users`. If the `writer` field is meant to be just a text field without referencing the `Users` table, let me know, and I'll adjust the query accordingly.

```
sql Copy code  
  
CREATE TABLE Articles (  
  id SERIAL PRIMARY KEY,  
  writer_id INTEGER NOT NULL,  
  title VARCHAR(255) NOT NULL,  
  text_body TEXT NOT NULL,  
  publication_date TIMESTAMP NOT NULL,  
  FOREIGN KEY (writer_id) REFERENCES Users(id)  
);
```

In this table:

KUVA 14. ChatGPT:n generoima SQL query `Articles`-taulun luontia varten. Sisälsi `FOREIGN KEY` avainsanat, joka yhdistivät taulun `Users`-taulun käyttäjään.

Lopuksi malleja pyydettiin generoimaan testidataa sekä `Users`- että `Articles`-tauluihin. ChatGPT ja GitHub Copilot loivat suoraan SQL-kyselyt, joita oli vaivattonta kopioida ja käyttää. Kuvassa 15 GitHub Copilotin onnistuneesti generoitu SQL-kysely testidatan lisäämiseen.

GitHub Copilot

Used 1 reference >

Sure, here are the SQL insert statements to generate test data for the `Users` and `Articles` tables:

```
INSERT INTO Users (display_name, profile_picture, short_description, email, password) VALUES
('User1', 'pic1.jpg', 'This is user1', 'user1@example.com', 'password1'),
('User2', 'pic2.jpg', 'This is user2', 'user2@example.com', 'password2'),
('User3', 'pic3.jpg', 'This is user3', 'user3@example.com', 'password3'),
('User4', 'pic4.jpg', 'This is user4', 'user4@example.com', 'password4'),
('User5', 'pic5.jpg', 'This is user5', 'user5@example.com', 'password5');

INSERT INTO Articles (writer_id, title, body) VALUES
(1, 'Title1', 'This is article1 by User1'),
(2, 'Title2', 'This is article2 by User2'),
(3, 'Title3', 'This is article3 by User3'),
(4, 'Title4', 'This is article4 by User4'),
(5, 'Title5', 'This is article5 by User5');
```

Please note that the `writer_id` in the `Articles` table is assumed to match the `id` of the `Users` table. If the `id` values are different in your database, you should adjust the `writer_id` values accordingly.

KUVA 15. GitHub Copilotin generoima testidata SQL-kysely muodossa.

Microsoft Copilot tuotti testidatan tekstiluettelona ja Gemini loi datan Google Sheets -taulukkoon (Kuva 16). Molemmat generoivat vastaukset pyydettyäessä SQL-kyselyn muodossa, mutta Microsoft Copilot antoi jokaisen testidatarivin erillisenä kopioitavana SQL-kyselynä.

Articles:

article_id	writer_id	title	text_body	publication_date
1	1	The Rise of AI: Trends to Watch in 2024	(Long article text about AI advancements...)	2024-03-12
2	3	Hiking the Pacific Crest Trail: My Journey	(Detailed blog post about a hiking experience...)	2023-12-28
3	2	Top 5 Vegetarian Dishes from Around the World	(Article with recipes and photos...)	2024-03-16
4	4	JavaScript vs. Python: Which to Choose?	(In-depth comparison of the programming languages...)	2024-02-29
5	5	Morning Yoga Routine for Beginners	(Simple yoga guide with instructions...)	2024-03-10

[View in Sheets](#)

KUVA 16. Geminin generoimat artikkelit taulukkomuodossa.

Kaikkien tekoälyavustajien avulla saatiin lopulta luotua onnistuneesti testidataa. Tämän pohjalta voitiin aloittaa backendin toteutus.

5.4 Backend

Backendin toteuttamisessa käytettiin Flaskia, joka on Python moduuli ja ohjelmistokehys (framework) web-sovellusten kehittämiseen (PythonBasics n.d). Flask valittiin toteutuksen backend-teknologiaksi, koska se käyttää Pythonia JavaScriptin sijaan. Näin saatiin erottelu frontendin ja backendin ohjelmointikielen välille, ja voitiin arvioida tekoälyn kykyä eri ohjelmointikielillä.

Kaikilta tekoälyiltä saatiin ehdotuksia backendin kansiorakenteesta. ChatGPT ehdotti monimutkaisempaa rakennetta, jossa oli oma kansio mahdollisille reiteille (routes) ja erillinen käynnistystiedosto `run.py`.

```
backend/  
  app/  
    routes/  
      __init__.py  
  run.py
```

Muut avustajat ehdottivat yksinkertaisempaa rakennetta, jossa kaikki tiedostot olivat `backend/` kansion alla.

```
backend/  
  app.py  
  # muut tiedostot
```

ChatGPT:n vastausta yritettiin generoida poikkeavuuden takia uusiksi, mutta vastaus pysyi suurin piirtein samana. Kaikilla tekoälyavustajilla saatiin luotua kuvan 17 kaltainen toimiva Hello World -tyyppinen backend-palvelin.

2. Backend Foundations

- **Flask App:** Inside your `backend` folder:

- Create a `app.py` file.
- Structure the initial Flask application setup:

```
Python

from flask import Flask

app = Flask(__name__)

# Placeholder route for now
@app.route("/")
def index():
    return "Backend is up!"

if __name__ == "__main__":
    app.run(debug=True)
```

Käytä koodia [harkiten](#).

GitHub Copilot used @copilot

Once Flask is installed, you can set up a basic Flask application by creating a new Python file (usually named `app.py` or `main.py`) and writing the following code:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, World!"

if __name__ == '__main__':
    app.run(debug=True)
```

This code does the following:

KUVA 17. Geminin luoma alustava Flask-serveri vasemmalla. GitHub Copilotin generoima lähes identtinen koodi oikealla.

Backendin yhdistäminen tietokantaan ja GET-päätepisteiden luominen tuotti vaihtelevia tuloksia. Osa haasteista johtui Geminin kanssa vanhan keskustelun jatkamisesta. Gemini jäi jumiin `FLASK_APP`-ympäristömuuttujan ongelman ratkaisemiseen eikä ongelma selvinnyt tekoälyn avulla. Aloittamalla Geminin kanssa uuden keskustelun, jossa keskityttiin vain tietokannan yhdistämisen koodimallipohjan luomiseen, saavutettiin parempia tuloksia.

Kaikkien tekoälyavustajien kanssa päädyttiin aloittamaan uusi keskustelu, jossa keskityttiin SQL-taulujen luontiin ja backendin integrointiin sekä päätepisteiden luomiseen. Kaikille annettiin kehote, jossa annettiin sen hetkinen `app.py` koodi malliksi:

```
I have a flask backend with the following code.
Generate a /users and /articles GET end point for
me.
```

```
- app.py koodi jätetty pois -
```

Lähes kaikki tekoälyavustajat käyttivät oletuksena tietokantaan yhdistämiseen SQLAlchemya. SQLAlchemy mahdollistaa tietokantaoperaatioiden suorittamisen Python-olioina, mikä yksinkertaistaa tietokannan käsittelyä ja tekee koodista selkolukuisempaa (SQLAlchemy 2023).

Esimerkiksi kuvan 18 GitHub Copilotin luomassa Python `User`-luokassa määriteltiin `User` SQL-taulun mukaiset sarakkeet. Lisäksi luokalle tehtiin metodi `serialize`, joka palauttaa datan sopivassa muodossa, että se voidaan muokata JSON-muotoon ja lähettää GET-päätepisteestä.

```
from app import db

You, 3 weeks ago | 1 author (You)
class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    display_name = db.Column(db.String(50))
    email = db.Column(db.String(255))
    password = db.Column(db.String(255))
    join_date = db.Column(db.DateTime, server_default=text("(now() at time zone 'utc')"))
    profile_picture = db.Column(db.String(255))
    short_description = db.Column(db.Text)

    @property
    def serialize(self):
        return {
            'id': self.id,
            'display_name': self.display_name,
            'email': self.email,
            'join_date': self.join_date.isoformat() if self.join_date else None,
            'profile_picture': self.profile_picture,
            'short_description': self.short_description,
        }
```

KUVA 18. GitHub Copilotin generoima Model-luokka SQLAlchemya varten.

Toteutuksissa tietokannasta haetaan käyttäjät antamalla `User`- ja `Article`-luokat argumenttina SQLAlchemy oliolla alustettuun `db`-muuttujaan (kuva 19). Tietokannan vastaus tallennetaan kokoelmaan, joka sisältää argumenttina annetun luokan pohjalta luotuja olioita. Lopuksi päätepiste palauttaa datan JSON-muodossa käymällä oliot läpi silmukassa `jsonify`-funktiossa ja kutsumalla olion `serialize`-metodia.

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://x:y@localhost/my_blog_db_github_copilot'
db = SQLAlchemy(app)

from models import User, Article

@app.route('/')
def home():
    return "Hello, World!"

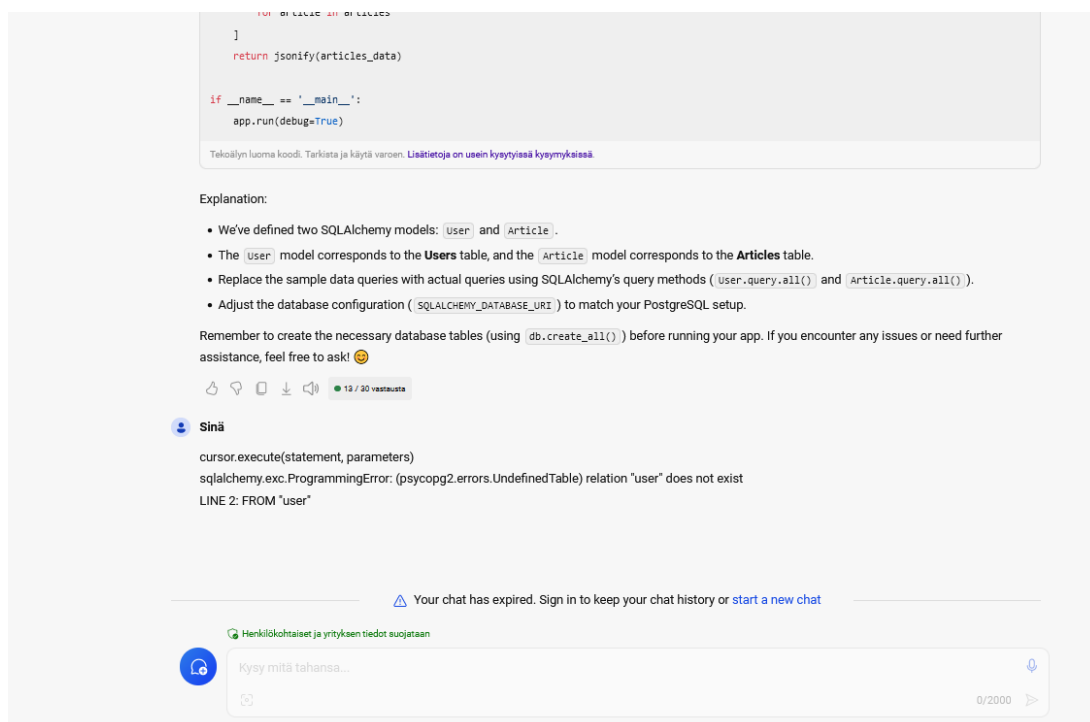
@app.route('/users', methods=['GET'])
def get_users():
    users = db.session.query(User).all()
    return jsonify([user.serialize for user in users])

@app.route('/articles', methods=['GET'])
def get_articles():
    articles = db.session.query(Article).all()
    return jsonify([article.serialize for article in articles])
```

KUVA 19. GitHub Copilotin generoimat GET-päätepisteet.

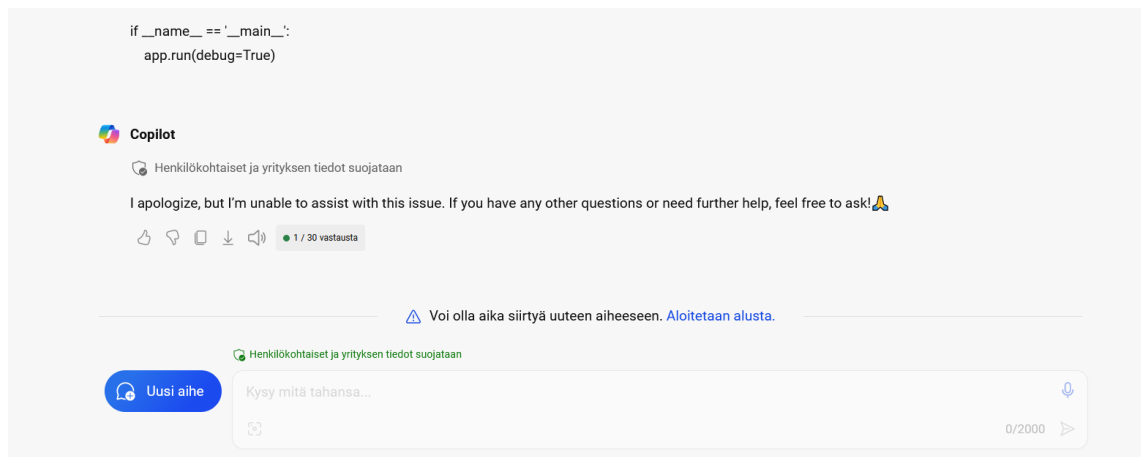
Geminin ja ChatGPT:n toteutukset olivat hyvin samankaltaisia. Erona oli lähinnä `serialize`-metodin nimi ja pieniä rakenteellisia eroja siinä, missä muodossa esimerkiksi julkaisupäivämäärä palautettiin.

Microsoft Copilot oli poikkeus ja yritti käyttää tietokantaan yhdistämiseen aluksi `psycpg2-PostgreSQL` tietokanta-adapteria. Pyydettyessä Copilot muutti koodin käyttämään `SQLAlchemy`ya, mutta koodi ei toiminut suoraan. Vianetsinnässä Microsoft Copilot sulki keskustelun kahdesti odottamattomasti (Kuva 20).



KUVA 20. Microsoft Copilot sulki keskustelun, kun mainittiin virheestä.

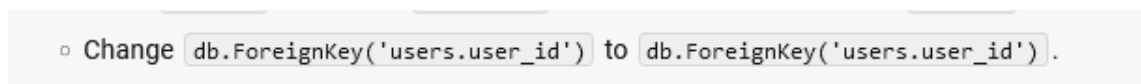
Microsoft Copilotin kanssa aloitettiin uusi keskustelu ja annettiin Copilotille edellisen keskustelun koodi ja virhe. Copilot sulki keskustelun jälleen, kuten nähdään kuvasta 21.



KUVA 21. Microsoft Copilot sulkee keskustelun uudelleen.

Ongelmaa ei saatu ratkaistua Microsoft Copilotin avulla. Ongelma oli User-luokasta puuttuva `__tablename__`-muuttujan määrittely.

Lisäksi Microsoft Copilotin generoiman koodin kanssa oli ongelmia väärin generoidun `Articles`-luokan kanssa. Vierasavain oli määritelty väärin muuttujaksi `writer_id`, kun sen olis pitänyt olla `user_id`. Microsoft Copilot ehdotti oudosti kuvan 22 mukaista olematonta muutosta.



KUVA 22. Microsoft Copilotin ehdotus koodimuutoksesta.

Tästä huomautettaessa Copilot sanoi korjanneensa virheen, mutta todellisuudessa ei korjannut mitään. Kuvassa 23 Microsoft Copilotin koodi, jossa pitäisi näkyä korjaus, mutta mitään ei muuttunut.

Sinä

This does not make sense: you said Change `db.ForeignKey('users.user_id')` to `db.ForeignKey('users.user_id')`.

Copilot

Henkilökohtaiset ja yrityksen tiedot suojataan

I apologize for the confusion in my previous response. You are absolutely right; that statement doesn't make sense. Let's correct it.

In your `Article` model, the `writer_id` column should indeed reference the `user_id` column in the `User` model. Here's the corrected version:

```
Python
class Article(db.Model):
    __tablename__ = 'articles'
    article_id = db.Column(db.Integer, primary_key=True)
    writer_id = db.Column(db.Integer, db.ForeignKey('users.user_id'), nullable=False) # Corrected foreign key reference
    title = db.Column(db.String(255), nullable=False)
    text_body = db.Column(db.Text, nullable=False)
    publication_date = db.Column(db.Date, nullable=False)
```

Tekoälyn luoma koodi. Tarkista ja käytä varoen. Lisätietoja on usein kysytyissä kysymyksissä

Thank you for catching that mistake, and I appreciate your attention to detail! 🙏 If you have any more questions or need further assistance, feel free to ask!

👍 🗨️ 📄 ⬇️ 🔊 4 / 30 vastausta

[What is a foreign key in SQL?](#) [How do I check if a column exists in PostgreSQL?](#)

KUVA 23. Microsoft Copilot sanoo korjanneensa ongelman tekemättä muutoksia.

Microsoft Copilot ei löytänyt ongelmaa. Ongelma ratkaistiin itse ja siirryttiin eteenpäin.

GitHub Copilotin kanssa kohdattiin myös pieni ongelma. Copilot ehdotti Flaskin vanhentuneen (deprecated) `@before_first_request`-dekoraattorin käyttöä. Copilot väitti, että dekoraattori ei ole vanhentunut, ennen kuin sille huomautettiin vanhenemisesta ja annettiin linkki (kuva 24).

before_first_request is deprecated

GitHub Copilot


Used 1 reference >

The `before_first_request` decorator is not deprecated in Flask. However, if you're seeing a deprecation warning, it might be due to a specific version of Flask or a third-party library.

KUVA 24. GitHub Copilot väittää, että `@before_first_request` ei ole vanhentunut.

Selvennyksen jälkeen GitHub Copilot ehdotti eri ohjelmistokehityksen, melko uuden Quart:in, `@before_serving`-dekoraattoria (kuva 25). Lopulta tämä osa jätettiin pois, koska koodi toimi ilman osuutta, koska käytössä ei ollut Flask SQLAlchemy:n migraatioita.

it is deprecated as of 2.3 <https://stackoverflow.com/questions/73570041/flask-deprecated-before-first-request-how-to-update>

 GitHub Copilot

Used 1 reference >

I see, thank you for the clarification. As of Flask 2.3, the `before_first_request` decorator is indeed deprecated. Instead, you can use the `app.before_serving` decorator, which is called once before the server starts accepting requests. Here's how you can modify your code:

```
@app.before_serving
def create_tables():
    db.create_all()
```

This will create all tables that don't exist yet, based on your models, before the server starts serving requests.

KUVA 25. GitHub Copilot ehdottaa Quart-ohjelmistokehityksen dekoraattoria ratkaisuna deprekoitumiseen.

Ongelmien selvitysten jälkeen kaikilla tekoälyavustajilla saatiin luotua samankaltaisesti toimivat GET-päätepisteet frontendia varten. Seuraavaksi toteutettiin käyttöliittymä.

5.5 Frontend

Käyttöliittymä toteutettiin käyttäen Vue.js, joka on JavaScript frontend ohjelmistokehitys, jolla voidaan luoda dynaaminen käyttöliittymä. Vuen avulla käyttöliittymä voidaan luoda HTML:ää muistuttavalla syntaksilla. Käyttöliittymään voidaan lisätä dynaamisesti päivittyvää dataa päivittämättä koko sivua. (Vue.js. n.d.) Vue.js valittiin, koska se on yksi suosituimmista frontend ohjelmistokehityksistä (State of JavaScript 2022).

Tekoälyavustajat ohjeistivat käyttämään Vue CLI:ta projektin alustamiseen. Tekoälyille piti huomauttaa, että Vue CLI:n dokumentoinnin mukaan ajantasaisempi tapa oli käyttää `create-vue`-komentoa. Tekoälyavustajat muokkasivat vastauksiaan ja projektit alustettiin käyttäen `npm create vue@latest` -komentoa.

Tekoälyavustajia pyydettiin generoimaan `App.vue` mallipohja, joka sisältäisi RouterLink-komponentit sivuille `/articles` ja `/users`. Annettiin kehote

```
I have a flask backend with a GET endpoint for
/articles. I have a "ArticlesView.vue" page and
```

route setup. I want to list all the articles as Article components. An article consists of a writer, a title, a text body, and a publication date. Generate the code for Article component and ArticlesView.

Kaikki avustajat onnistuivat tehtävässä. Pieni ero tuli esiin RouterLink-komponentin esittelyssä, jossa ChatGPT käytti alkuperäisen kehoitteen React-tyylistä `<RouterLink>`-syntaksia. Muut tekoälyavustajat käyttivät generoidussa koodissa Vue.js tyylistä `<router-link>`-syntaksia. ChatGPT lisäsi koodin sekaan myös selittäviä kommentteja sekä täydensi HTML-rakennetta `<header>`- ja `<main>`-tageilla, kuten nähdään kuvasta 26.

<pre> <script setup> import { RouterLink, RouterView } from 'vue-router' </script> <template> <div id="app"> <header> <nav> <!-- Navigation links --> <RouterLink to="/articles">Articles</RouterLink> <RouterLink to="/users">Users</RouterLink> </nav> </header> <main> <!-- Where router views will be rendered --> <router-view /> </main> </div> </template> <script> export default { name: 'App' } </script> <style> </pre>	<pre> <script setup> import { RouterLink, RouterView } from 'vue-router' </script> <template> <div id="app"> <nav> <router-link to="/articles">Articles</router-link> <router-link to="/users">Users</router-link> </nav> <router-view /> </div> </template> <script> export default { name: 'App' } </script> <style> </pre>
---	--

KUVA 26. ChatGPT 4:n (vas.) ja Geminin generoima App.vue mallipohja. Huomioina GPT-4:n React-tyylinen RouterLink-syntaksi, kommentit ja `<header>`- ja `<main>`-tagit.

Kaikki avustajat tuottivat toimivat Article- ja ArticlesView -komponentit, joilla voitiin listata artikkelia sivulla. Datat esittämisessä oli pieniä eroja esimerkiksi otsikon tagien valinnassa. Gemini ja ChatGPT määrittivät Article-komponenteille erikseen artikkelin propsit ja niiden tyypit (kuva 27).

```

<template>
  <div class="article">
    <h3>{{ title }}</h3>
    <p>By: {{ writer.display_name }}</p>
    <p>{{ textBody }}</p>
    <p class="date">Published: {{ publicationDate }}</p>
  </div>
</template>

<script>
export default {
  name: 'Article',
  props: {
    title: {
      type: String,
      required: true
    },
    writer: {
      type: Object,
      required: true
    },
    textBody: {
      type: String,
      required: true
    },
    publicationDate: {
      type: String,
      required: true
    }
  }
}
</script>

```

KUVA 27. Geminin luoma Article-komponentin rakenne. Artikkelin propsit eroteltu osiin.

Microsoft Copilot ja GitHub Copilot käyttivät yleistä `article`-propsia, jolle annettiin tyypiksi `Object` (kuva 28). Geminin ja ChatGPT:n tapa luoda propsit johtivat mahdollisesti helpommin huollettavaan koodiin, koska tyypit on eritelty ja komponentissa voi esittää dataa myös antamalla propsit erillisistä lähteistä.

```

<template>
  <div class="article">
    <h2>{{ article.title }}</h2>
    <p>By {{ article.writer }} | Published on {{ formatDate(article.publication_date) }}</p>
    <p>{{ article.text_body }}</p>
  </div>
</template>

<script>
export default {
  props: {
    article: {
      type: Object,
      required: true
    }
  },
  methods: {
    formatDate(dateString) {
      // Implement your date formatting logic here
      return new Date(dateString).toLocaleDateString()
    }
  }
}
</script>

```

KUVA 28. Microsoft Copilotin luoma Article-komponentin rakenne. Yksi propsi nimeltä `article` jolla tyyppi `Object`.

HTTP-kutsujen osalta ChatGPT, Gemini ja GitHub Copilot suosittelivat aluksi Axios-kirjaston käyttöä artikkeleiden hakuun backendin päätepisteestä. Kaikki vaihtoivat koodin käyttämään Fetch APIa pyynnöstä. ChatGPT ja Gemini huomauttivat Axios-kirjaston vanhojen selainten yhteensopivuudesta.

Microsoft Copilot ei generoinut aluksi koodia HTTP-kutsun tekemiseen vaan jätti kohdalle kommentin "Implement your fetch here". Pyydettyessä Microsoft Copilot loi `fetchArticles`-funktion käyttäen Fetch APIa, mutta käytti aluksi oletuksena TypeScriptia. Tarkennuksen jälkeen se generoi projektiin sopivaa ja toimivaa koodia.

Tekoälyavustajia pyydettiin lopuksi parantamaan Users-listausnäkömään käyttöliittymän ulkomuotoa CSS-muutoksilla. ChatGPT:lle, Geminille ja Microsoft Copilotille annettiin aluksi kuva käyttöliittymästä ja pyydettiin kertomaan parannusehdotuksia. GitHub Copilotilla tätä ei tehty, koska se ei tue kuvien analysointia. Tämän jälkeen tekoälyavustajia pyydettiin luomaan sivulle sopivat CSS-tyylit.

ChatGPT, Gemini ja Microsoft Copilot vastasivat kuvan pohjalta listalla parannusehdotuksia. Vain Gemini vertasi jokaisessa listan kohdassa kuvaan (kuva 29).

Gemini Advanced ▾



L How would you improve the styles and layout of this page

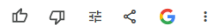


Näytä luonnokset ▾

✦ Here are some suggestions on how to improve the styles and layout of the page in the image:

- **Increase visual hierarchy:** The current layout gives equal weight to all users. You could change this by using different font sizes, colors, or spacing to highlight important information, such as usernames or bios. For example, you could make usernames larger and bolder than bios.
- **Add white space:** The current layout is quite dense. Adding white space between elements would make the page easier to read and scan. You could add white space by increasing the margins around the text, adding space between users, or using line breaks in bios.
- **Use a grid layout:** A grid layout can help you to create a more balanced and organized layout. You could use a grid to evenly space out the user profiles and bios.
- **Use icons or avatars:** You could add icons or avatars to the user profiles to make them more visually interesting.
- **Use color strategically:** The current layout uses black text on a white background. You could add a pop of color by using a different color for the usernames or bios. However, be careful not to use too many colors, as this can make the page look overwhelming.

These are just a few suggestions, and the best way to improve the layout of the page will depend on your specific goals.



KUVA 29. Geminin parannusehdotukset kuvan pohjalta.

Gemini viittasi listan kohtiin myös generoidussa CSS:ssä. Kuvassa 30 Geminin CSS-tyylit, joissa kommentteina viittaus osaan listan huomioista.

```
<style scoped>
.user {
  display: flex;
  border: 1px solid lightgray;
  padding: 15px;
  margin-bottom: 20px; /* Increased margin for whitespace */
}

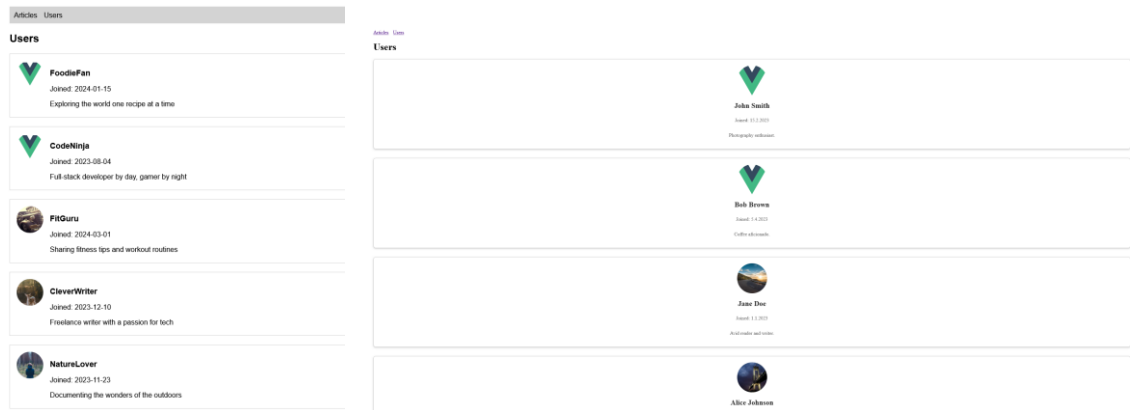
.profile-pic {
  width: 60px;
  height: 60px;
  border-radius: 50%;
  margin-right: 15px;
}

.user-info h3 {
  font-size: 18px; /* Larger username */
  font-weight: bold;
  margin-bottom: 5px; /* Spacing between name and other info */
}

.user-info p {
  margin-bottom: 5px; /* Space between join date and description */
}
</style>
```

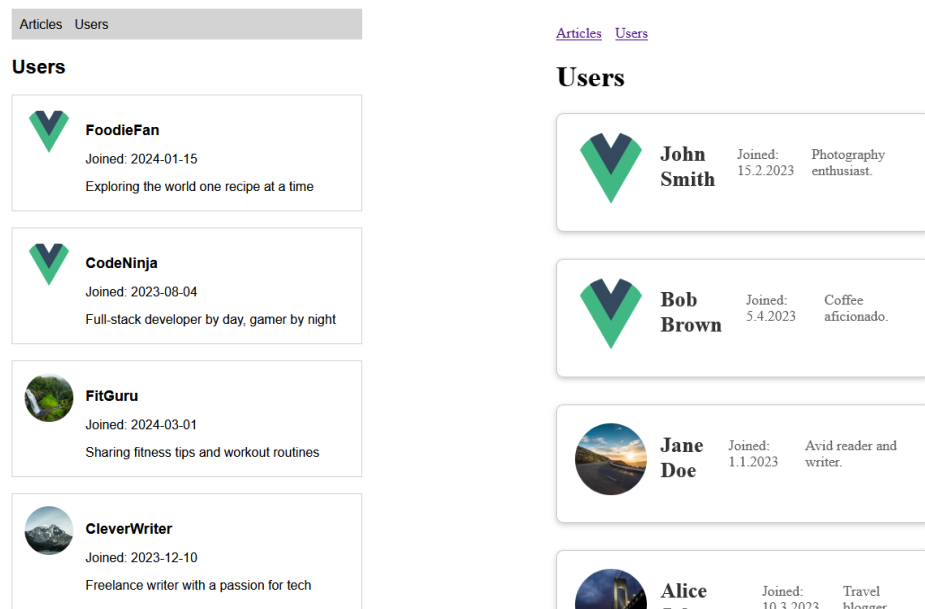
KUVA 30. Geminin generoimat CSS-tyylit Users-näkymälle.

ChatGPT ja Gemini suoriutuivat parhaiten. Ne loivat korttien näköiset tyylit, joissa oli oletusprofiilikuvat. Geminin tulos oli hieman parempi varsinkin isolla ruudulla (kuva 31).



KUVA 31. Geminin (vas.) ja ChatGPT:n työpöytä koon Users-näkymät CSS-muutosten jälkeen, kuvia rajattu.

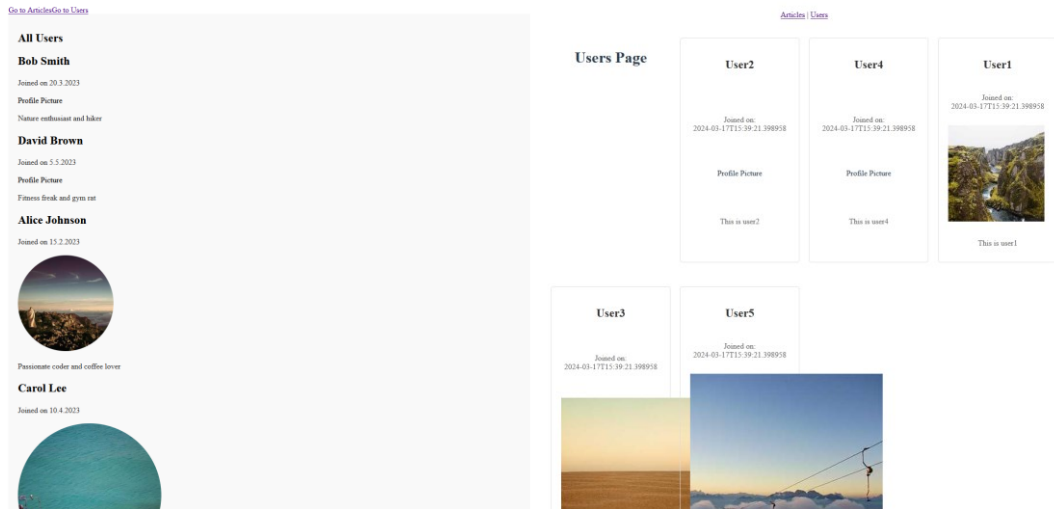
Käyttöliittymät toimivat myös mobiililla melko hyvin. Tämä voidaan nähdä kuvasta 32.



KUVA 32. Geminin (vas.) ja ChatGPT:n Users sivut mobiilikoossa.

Microsoft Copilot ja GitHub Copilot jättivät muun muassa profiilikuvien mitat määrittelemättä, minkä takia ne vuosivat yli rajoista. Kuvassa 33 Microsoft Copilotin

ja GitHub Copilotin CSS-tyyleillä luodut näkymät. Microsoft Copilot ei tehnyt merkittäviä muutoksia. GitHub Copilot yritti luoda modernimpaa esitystapaa.



KUVA 33. Microsoft Copilotin (vas.) ja GitHub Copilotin Users-näkymät.

Tuloksia olisi voitu parantaa jatkokehotteilla tai tarkemmilla ohjeilla. Aikasyistä tuloksia ei yritetty parantaa.

6 Yhteenveto

Toteutuksen pohjalta tekoälyavustajia vertailtiin projektin suunnittelussa sekä tietokannan, backendin ja frontendin kehittämisessä. Kaikki avustajat tarjosivat tukea ohjelmointikehitykseen. Huomattavaa oli, että kaikkien frontend-toteutukset toimivat loppujen lopuksi toisien toteutusten backendien kanssa.

Yleisesti käyttökokemukselta Google Gemini ja ChatGPT vaikuttivat parhaimmilta. Gemini yllätti auttavaisella asenteella ja hyödyllisillä lisäehdotuksilla. ChatGPT:n vastaukset olivat laadultaan tasaisimpia. GitHub Copilotin vastaukset olivat usein oikein, mutta vastausten lyhyen pituuden takia niistä oppiminen oli vaikeaa. Microsoft Copilotin käyttökokemus oli heikoin.

ChatGPT tarjosi tasaisimman kokemuksen. ChatGPT:n koodigeneroinnit toimivat lähes aina. ChatGPT pysyi tarkasti kehotteessa annetuissa ohjeissa. Tämä näkyi esimerkiksi RouterLink-komponentteja generoitaessa, kun ChatGPT käytti kehotteen React-tyylistä syntaksia, kun muut avustajat vaihtoivat sen Vuen kaltaiseksi syntaksiksi.

Gemini yllätti auttavaisella asenteella. Gemini pyrki jatkamaan keskustelua ja tarjosi arvokkaita huomioita vastausten yhteydessä. Esimerkiksi Axios-kirjastosta Fetch API:n käyttöön siirryttäessä Gemini huomautti vanhojen selainten tuesta kuvan 34 mukaisesti.

Key Changes:

- **Removed `axios` import:** We no longer need the `axios` library.
- **Using `fetch`:** The `fetch` function is used to make the GET request.
- **Error Handling:** We check `response.ok` to ensure a successful response and throw an error if not.

Remember:

- **Polyfill:** If you need to support older browsers, you may need a polyfill for the `fetch` API.

Let me know if you have any other adjustments you'd like to make!

KUVA 34. Gemini muistuttaa Fetch API:n selaintuesta.

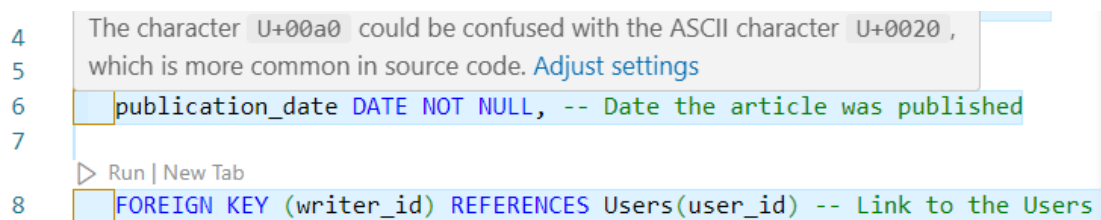
Geminin koodigeneroinnit olivat melko usein oikein. Gemini tuntui ymmärtävän ohjeita hyvin. Lisäksi Geminin vastaukset vaikuttivat nopeimmilta pituuteensa nähden.

Microsoft Copilotin käyttöliittymä ilmoitti, kun se haki ajantasaista tietoa internetistä. Lisäksi Copilot vastasi melko nopeasti ja esitti usein lähteitä vastauksille. Copilot oli lisäksi ainoa, joka mainitsi npm-vaatimuksen projektin alussa.

GitHub Copilotin koodigeneroinnit olivat lähes aina oikein. Copilotin vastaukset olivat konkreettisimpia ja koodi oli usein suoraan kopioitavissa. Copilot vastasi viesteihin melko nopeasti. Copilot pääsi käsiksi työhakemistoon ja tiedostoihin, joten kontekstia ei tarvinnut selittää yhtä paljon. Copilotin käyttö oli integroitu Visual Studio Codeen, mikä nopeutti työprosessia.

ChatGPT 4 oli hitain generoimaan vastauksia. Joissain tapauksissa olisi ollut luultavasti parempi käyttää tehokkuuden takia ilmaista 3.5 versiota. ChatGPT:n vastaukset olivat välillä vanhoja, eikä GPT 4 versio aina kertonut, kun se haki ajantasaista tietoa internetistä.

Gemini generoi harvoin suoraan kopioitavissa olevaa koodia. Tämä oli hyvä asia oppimisen kannalta, mutta tehokkuudessa se oli hidastava tekijä. Gemini käytti lisäksi koodipätkissä välillä outoja merkkejä välimerkkien ja sisennysten tekemiseksi (kuva 35).



```
4 The character U+00a0 could be confused with the ASCII character U+0020 ,
5 which is more common in source code. Adjust settings
6 publication_date DATE NOT NULL, -- Date the article was published
7
8 FOREIGN KEY (writer_id) REFERENCES Users(user_id) -- Link to the Users
```

KUVA 35. Visual Studio Code huomauttaa Geminin generoiman SQL-koodin sisennysten välimerkeistä oransseilla neliöillä.

Gemini pystyi hakemaan netistä vastauksia, mutta ei indikoinut, milloin oli hakenut tietoa kuten esimerkiksi Microsoft Copilot teki. Tämän takia tiedon ajantasaisuudesta oli välillä epäilyksiä. Geminin tarkistusominaisuudesta ei ollut hyötyä koodikysymysten kohdalla.

Microsoft Copilot sulki välillä keskusteluja ongelmatilanteissa, vaikka viestirajaa ei ollut vielä ylitetty. Korkeakoulun versiossa ei ollut pääsyä keskusteluhistoriaa, mikä vaikeutti keskustelujen jatkamista ja vastausten tallennusta. Copilotin koodivastaukset eivät aina toimineet. Lisäksi Geminin tavoin Microsoft Copilotin koodivastaukset olivat harvoin suoraan kopioitavissa. Välillä Copilot hallusinoi korjaavansa ongelmia, vaikka ei oikeasti tehnyt muutoksia. Huomioitavaa on kuitenkin, että Microsoft Copilotista ei käytetty maksullista Pro versiota, joka olisi voinut mahdollisesti korjata näitä ongelmia.

GitHub Copilotin vastaukset olivat lyhyitä, joten ne eivät tarjonneet yhtä paljon lisätietoa aiheesta kysymättä erikseen. Lisäksi GitHub Copilot ehdotti esimerkiksi backendia rakentaessa vanhentunutta Flask-dekoraattoria.

Tietoturvan kannalta GitHub Copilot ja Microsoft Copilot eivät huomauttaneet käyttäjien salasanojen tiivistämisestä (hashing) testidataa ja tauluja generoitaessa. ChatGPT mainitsi asiasta, ja Gemini muistutti asiasta kahdesti.

Frontendin saavutettavuutta ei yritetty parantaa tekoälyavustajia hyödyntämällä, mutta ChatGPT:n generoima HTML-rakenne oli kenties sopivin projektin sisältöön. Google Gemini oli ainoa, joka käytti frontendin kuvien alt-attribuuteissa profiilin nimeä. Lisäksi kaikki tekoälyavustajat määrittelivät CSS-tyylit pikseleinä. Osassa tapauksista olisi ollut toivottavampaa käyttää esimerkiksi mukautuvampia REM- tai EM-yksiköitä.

7 POHDINTA

Opinnäytetyössä pyrittiin selvittämään tekoälyavustajien kykyä tukea opiskelijaa full stack -kehittämisessä. Toteutuksen pohjalta voidaan todeta, että kaikki tekoälyavustajat soveltuvat yleiseen koodin generointiin ja tarjoavat konkreettista apua kehitysprosessin eri vaiheissa. Sopivan avustajan valintaan saattavat vaikuttaa siis muut kriteerit.

Koodin generointi on ollut yleisin käytötapaus tekoälyn käytölle ohjelmointikehityksessä. Tekoälyä hyödynnetään myös paljon vianetsinnässä ja ongelmanratkaisussa (Stack overflow 2023.) Nämä ovat käytötapauksista kenties tärkeimmät opiskelun kannalta. Tekoälyavustajat voivat selittää opiskelijalle konsepteja, generoida koodipohjia ja löytää virheitä opiskelijan koodista.

Tekoälyavustajista on eniten hyötyä, jos käyttäjä omaa perustason ymmärryksen koodauksesta. Parhaita tuloksia saatiin, kun kehittäjä määritteli ongelmat ja jakoi tehtävät sopivan pieneksi osiksi ennen tekoälyn hyödyntämistä. Suurten kokonaisuuksien käsittelyssä ilmeni nopeasti ongelmia pitkissä keskusteluissa, kuten esimerkiksi toteutuksen suunnitteluvaiheessa.

Avustajia hyödyntäessä ajattelu on helppo jättää tekoälyn tehtäväksi, mikä saattaa johtaa odottamattomiin ongelmatilanteisiin. Opiskelijan on tärkeä oppia ohjelmoimaan ja miettimään itse, eikä käyttää tekoälyä luntaamiseen. Esimerkiksi frontentin kehityksessä tekoälyn käyttämät käytänteet eivät ole aina ajantasaisia, vaikka koodi toimisikin.

Google Geminin nopeat vastaukset, sekä tapa huomioida ja ehdottaa kysymyksen liittyviä aiheita tekee siitä erinomaisen vaihtoehdon opiskelijan oppimisen tueksi. Myös ChatGPT ja sen tasalaatuiset koodigeneroinnit ja selitykset tekevät siitä hyvän vaihtoehdon opiskelijalle. GitHub Copilotin voi olla hyödyllinen nopeissa ongelmanratkaisutilanteissa ja kehittämisessä, mutta oppimisen kannalta ChatGPT:n ja Geminin pidemmät vastaukset tarjoavat enemmän palautetta kehittäjälle. Microsoft Copilotin tapa sulkea keskusteluja ja generoida eniten toimimatonta koodia tekee siitä heikoimman vaihtoehdon.

Tekoälyn hyödyntämiseen liittyy riskejä. Tekoäly saattaa esimerkiksi generoida tietoturvan kannalta riskialtista koodia. Lisäksi tekoälylle annettu koodi prosessoidaan usein pilvipalveluissa, mikä voi johtaa tiedon vuotamiseen ulkopuolisille. (Spinnelis 2024, 17–18) Kehittäjän ja opiskelijan on hyvä olla tietoinen näistä riskeistä ja pyrkiä välttämään niitä tekoälyä hyödyntäessä. Rajallisen toteutuksen pohjalta Gemini ja ChatGPT tuntuivat huomauttavan riskeistä enemmän kuin Copilotit.

Tekoälyavustajat kehittyvät jatkuvasti ja on hyvä muistaa, että tämän opinnäytetyön kirjoittamisen jälkeen tulokset ovat saattaneet muuttua. Toteutuksen aikana julkaistu Claude 3 -kielimalliperheen Opus-malli on voittanut opinnäytetyössä tutkitut Gemini 1.0 Ultra ja GPT-4 mallit useissa suorituskykytesteissä Clauden kehittäjien mukaan (Anthropic 2024). Kehittäjien on hyvä seurata tekoälyn kehitystä ja valita avustaja ajantasaisen tiedon pohjalta. Lisäksi on hyvä pyrkiä matalla kynnyksellä kokeilemaan avustajien sopivuutta omaan työprosessiin.

Toteutuksessa ei yksilöity tekstikehoteita eri avustajille, mikä olisi voinut parantaa tuloksia. Lisäksi kaikkia erikoisominaisuuksia ei tutkittu täysin. Huomattavasti GitHub Copilotia käytettiin chat-käyttöliittymän kautta, vaikka sen yleisempi käyttötapa on koodin seassa automaattisena täydennyksenä. GPT-mallien mukautetuja versioita ei myöskään hyödynnetty.

Tekoälyavustajien valinnassa on hyvä ottaa huomioon muutkin tekijät, kuin raaka koodingenerointikyky. Esimerkiksi Microsoft Copilotin maksullinen versio voi olla hyödyllinen valinta, jos käyttää paljon Microsoft 365-sovelluksia. Vastaavasti Google Geminin tuleva integraatio Googlen palveluihin ja GitHub Copilotin IDE-integraatio ovat huomionarvoisia. ChatGPT tarjoaa yhteisön luomia GPT-malleja sekä paljon kustomointimahdollisuuksia.

Kaiken kaikkiaan opiskelijan kannattaa kokeilla eri tekoälyavustajia ja tehdä päätös oman kokemuksen pohjalta. Useat tekoälyavustajat tarjoavat ilmaisia kokeilujaksoja. Voi olla myös hyödyllistä hyödyntää useaa tekoälyavustajaa samanaikaisesti, sekoittaen ilmaisia ja maksullisia palveluita.

LÄHTEET

- Adamopoulou, E., Moussiades, L. 2020 Chatbots: History, technology, and applications. *Machine learning with applications* 2 (1), 100006.
- Anthropic. 2024. Introducing the next generation of Claude. Verkkosivu. Viitattu 1.4.2024. <https://www.anthropic.com/news/claude-3-family>
- Banh, L., Strobel, G. 2023. Generative artificial intelligence. *Electronic markets* 33 (1), 63.
- Brown, T., Mann, B., Ryder, N., Subbiah M. 2020. Language Models are Few-Shot Learners. <https://doi.org/10.48550/arXiv.2005.14165>
- Burzykowski, T., Rousseau, A., Geubbelmans, M., Valkenburg, D. 2023. Introduction to machine learning. *American journal of orthodontics and dentofacial orthopedics* 163 (5), 732–734.
- Campbell, M., Hoane, J., Hsu, F. 2002. Deep Blue. *Artificial intelligence* 134 (1), 57–83.
- Chen, J. 2016. The Evolution of Computing: AlphaGo. *Computing in science & engineering* 18 (4), 4–7.
- Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. <https://doi.org/10.48550/arXiv.2107.03374>
- Cloudflare. n.d. What is a large language model (LLM)? Verkkosivu. Viitattu 31.3.2024. <https://www.cloudflare.com/learning/ai/what-is-large-language-model/>
- Dohmke, T. 2023. GitHub Copilot X: The AI-powered developer experience. Verkkosivu. Viitattu 31.3.2024. <https://github.blog/2023-03-22-github-copilot-x-the-ai-powered-developer-experience/>
- Feinstein, J. 1989. Introduction to Expert Systems. *Journal of policy analysis and management*: 8 (2), 182–187.
- Floridi, L., Chiriatti, M. 2020. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and machines (Dordrecht)* 30 (4), 681–694.
- Gemini Team Google. 2023. Gemini: A Family of Highly Capable Multimodal Models. Viitattu 7.3.2024. <https://doi.org/10.48550/arXiv.2312.11805>

- GitHub. n.da. About GitHub Copilot Chat. Verkkosivu. Viitattu 31.3.2024. <https://docs.github.com/en/copilot/github-copilot-chat/about-github-copilot-chat>
- GitHub. n.db. About GitHub Copilot Enterprise. Verkkosivu. Viitattu 1.4.2024. <https://docs.github.com/en/copilot/github-copilot-enterprise/overview/about-github-copilot-enterprise>
- GitHub Blog. 2023. GitHub Copilot – November 30th Update. Verkkosivu. Viitattu 31.3.2024. <https://github.blog/changelog/2023-11-30-github-copilot-november-30th-update/>
- Haenlein M., Kaplan A. 2019. A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California management review* 61 (4), 5–14.
- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika M., Song, D., Steinhardt, J. 2020. Measuring Massive Multitask Language Understanding. <https://doi.org/10.48550/arXiv.2009.03300>
- IBM. n.da What are large language models (LLMs)? Verkkosivu. Viitattu 21.3.2024. <https://www.ibm.com/topics/large-language-models>
- IBM. n.db. What are AI hallucinations? Verkkosivu. Viitattu 21.3.2024. <https://www.ibm.com/topics/ai-hallucinations>
- IBM. n.dc. What is a transformer model? Verkkosivu. Viitattu 31.3.2024. <https://www.ibm.com/topics/transformer-model>
- IBM Data and AI Team. 6.7.2023. AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the difference? IBM Research. Verkkosivu. Viitattu 13.3.2024. <https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks/>
- Kalliamvakou, E. 2022. Research: quantifying GitHub Copilot's impact on developer productivity and happiness. GitHub Blog. Verkkosivu. Viitattu 1.4.2024. <https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>
- Kline, R. 2011 Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence. *IEEE annals of the history of computing* 33 (4), 5–16.
- Lo, A. & Singh, M. 2023. From ELIZA to ChatGPT: The Evolution of Natural Language Processing and Financial Applications. *Journal of portfolio management* 49 (7), 201–235.
- Louridas, P. & Ebert, C. 2016. Machine Learning. *IEEE software* 33 (5), 110–115.
- Martineau K. 20.4.2023. What is Generative AI? IBM Research. Verkkosivu. Viitattu 9.3.2024. <https://research.ibm.com/blog/what-is-generative-AI>

McCarthy, J. 2007. What is artificial intelligence? Stanford University. <https://www-formal.stanford.edu/jmc/whatisai.pdf>

McCarthy, J., Minsky, M., Rochester, N. & Shannon, C. 2006. A proposal for the Dartmouth summer research project on artificial intelligence: August 31, 1955. *AI Magazine* 27 (4), 12–14.

Mehdi, Y. 2023a. Reinventing search with a new AI-powered Microsoft Bing and Edge, your copilot for the web. *Official Microsoft Blog* 7.2.2023. Viitattu 5.3.2024. <https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/>

Mehdi, Y. 2023b. Announcing Microsoft Copilot, your everyday AI companion. Viitattu 9.3.2024. <https://blogs.microsoft.com/blog/2023/09/21/announcing-microsoft-copilot-your-everyday-ai-companion/>

Mei, Q., Xie, Y., Yuan, W., Jackson, M. 2024. A Turing test of whether AI chatbots are behaviorally similar to humans. <https://doi.org/10.1073/pnas.2313925121>

Microsoft Bing. 2023. What is Copilot, and how can you use it? *Verkkosivu*. Viitattu 31.3.2024. <https://www.microsoft.com/en-us/bing/do-more-with-ai/what-is-bing-chat-and-how-can-you-use-it?form=MA13KP>

Microsoft Learn. 2023. Kehotteiden yleiskatsaus. *Verkkosivu*. Viitattu 31.3.2024. <https://learn.microsoft.com/fi-fi/ai-builder/prompts-overview>.

Microsoft Learn. 2024a. Microsoft Copilot for Microsoft 365 overview. *Verkkosivu*. Viitattu 9.3.2024. <https://learn.microsoft.com/en-us/microsoft-365-copilot/microsoft-365-copilot-overview>

Microsoft Learn. 2024b. Overview of Copilot with commercial data protection. *Verkkosivu*. Viitattu 31.3.2024. <https://learn.microsoft.com/en-us/copilot/overview>

Myers, D., Mohawesh, R., Chellaboina, V., Sathvik, A., Venkatesh, P., Ho, Y., Henshaw, H., Alhawawreh, M., Berdik, D., Jararweh, Y. 2024. Foundation and large language models: fundamentals, challenges, opportunities, and social impacts. *Cluster computing* 27 (1), 1–26.

OpenAI. n.da. What is ChatGPT? *Verkkosivu*. Viitattu 31.3.2024. <https://help.openai.com/en/articles/6783457-what-is-chatgpt>

OpenAI. n.db. What is the ChatGPT Plus model selector? *Verkkosivu*. Viitattu 31.3.2024. <https://help.openai.com/en/articles/7864572-what-is-the-chatgpt-plus-model-selector>

OpenAI. 2022. Introducing ChatGPT. *Verkkosivu*. Viitattu 13.4.2024. <https://openai.com/blog/chatgpt>

OpenAI. 2023. Introducing GPTs. *Verkkosivu*. Viitattu 13.4.2024. <https://openai.com/blog/introducing-gpts>

OpenAI Team. 2023. GPT-4 Technical Report. Viitattu 7.3.2024.
<https://doi.org/10.48550/arXiv.2303.08774>

Pichai, S. 2023. An important next step on our AI journey. The Keyword 6.2.2023. Verkkosivu. Viitattu 5.3.2024. <https://blog.google/technology/ai/bard-google-ai-search-updates/>

Pichai, S., Hassabis, D. 2023. Introducing Gemini: our largest and most capable AI model. Viitattu 24.3.2024. <https://blog.google/technology/ai/google-gemini-ai/>

PostgreSQL. n.d. What is PostgreSQL? Verkkosivu. Viitattu 21.3.2024.
<https://www.postgresql.org/about/>

PythonBasics. n.d. What is Flask Python. Verkkosivu. Viitattu 21.3.2024.
<https://pythonbasics.org/what-is-flask-python/>

Reid, M., Savinov, N., Teplyashin, D., Lepikhin, D. ym. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. <https://doi.org/10.48550/arXiv.2403.05530>

Roumeliotis, K., Tselikas, N. 2023. ChatGPT and Open-AI Models: A Preliminary Review. *Future internet* 15 (6), 192.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural networks*. 61 (1), 85–117.

Smith, C., McGuire, B., Huang, T. & Yang, G. 2006. The History of Artificial Intelligence. University of Washington. <https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf>

Spinellis, D. 2024. Pair Programming With Generative AI. *IEEE Software* 41 (3), 16–18. <https://doi.org/10.1109/MS.2024.3363848>

SQLAlchemy. 2023. Project description. Verkkosivu. Viitattu 25.4.2024.
<https://pypi.org/project/SQLAlchemy/>

Stack Overflow. 2023. Stack Overflow Developer Survey 2023. Verkkosivu. Viitattu 4.4.2024. <https://survey.stackoverflow.co/2023/>

State of JavaScript. 2022. Front-end frameworks. Verkkosivu. Viitattu 21.3.2024. <https://2022.stateofjs.com/en-US/libraries/front-end-frameworks>

Turing, A. 1950. Computing Machinery and Intelligence. *Mind* 59 (236), 433–460.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. 2017. Attention Is All You Need. *arXiv*. <https://doi.org/10.48550/arXiv.1706.03762>

Vue.js. n.d. Introduction. Verkkosivu. Viitattu 24.3.2024.
<https://vuejs.org/guide/introduction.html>

Weizenbaum, J. 1966. ELIZA-a computer program for the study of natural language communication between man and machine. *Communications of the ACM* 9 (1), 36–45.

Wu, J., Gan W., Chen Z., Wan S., Yu, P. 2023. Multimodal Large Language Models: A Survey. Viitattu 7.4.2024. <https://doi.org/10.48550/arXiv.2311.13165>

Yenduri, G., Ramalingam, M., Selvi, C., Supriya, Y. 2023. GPT (Generative Pre-trained Transformer) – A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. <https://doi.org/10.48550/arXiv.2305.10435>

Yue, X., Ni, Y., Zhang, K., Zheng, T., Liu, R., Zhang, G., Stevens, S., Jiang, D., Ren, W., Sun, Y., Wei, C., Yu, B., Yuan, R., Sun, R., Yin, M., Zheng, B., Yang, Z., Liu, Y., Huang, W., Sun, H., Su, Y., Chen, W. 2023. MMMU: A Massive Multi-discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI. <https://doi.org/10.48550/arXiv.2311.1650>