Tampereen ammattikorkeakoulu

Suuret kielimallit pelinkehityk-sessä

Samu Pulli

OPINNÄYTETYÖ
Toukokuu 2024

Tietojenkäsittelyn tutkinto-ohjelma
Games Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Games Production

Samu Pulli
Suuret kielimallit pelinkehityksessä

Opinnäytetyö 33 sivua
Toukokuu 2024

Opinnäytetyön tavoitteena oli tutkia suuria kielimalleja ja niiden toiminnan tehostamista RAG-dokumenttihaun avulla. Työ toteutettiin ajamalla tekoälymallit Docker-virtualisointiympäristössä omalla laitteistolla.

Työ aloitettiin vektorisoimalla Unity-pelimoottorin dokumentaatio, joka tallennettiin tietokantaan. Dokumentaatio siivottiin ja pilkottiin alikansioihinsa skriptillä, jonka jälkeen sen vektorisointi aloitettiin. Syötetty dokumentaatio sisälsi HTML-tiedostoja, jotka koostuivat käyttöohjeista ja koodiesimerkeistä. Tietokannasta dokumentit tulivat tekoälyn ja RAG-dokumenttihaun käytettäväksi.

Vektoritietokannan käyttäjäksi rakennettiin Kielimallin ja RAG:n yhdistelmä, jossa kaikki kolme järjestelmää voivat käyttää toisiaan tehostamaan ja parantamaan tuloksia. Käyttäjä voi valita järjestelmästä yksittäisen mallin tai käyttää niitä toistensa tukena. Käytetyt tekoälymallit valittiin avoimista vaihtoehdoista, jotta työn yksityisyystavoitteet täyttyisivät.

Valmistunutta järjestelmää hyödynnettiin ratkaisemaan vanhan pelinkehitysprojektin aiempia ongelmia. Testejä toteutettiin kaikilla malliyhdistelmillä, jotta tulosten eroavaisuudet selviäisivät.

Järjestelmä suoriutui tehtävistään ja sen toiminta parani, kun RAG-dokumenttihakua käytettiin kielimallin apuna. Lisäksi RAG suoriutui tietyistä tehtävistä paremmin kuin erillinen kielimalli.

Asiasanat: kielimalli, tekoäly, RAG

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Games Production

Samu Pulli:
Large Language Models in Game Development

Bachelor's thesis 33 pages
May 2024

The purpose of this thesis was to research and test Large Language Models as an assistant to game development. It further focused on Retrieval-Augmented Generation and how that can be used to improve language model's performance. Related models were also run on a local environment within Docker virtualization software.

The initial step in creating a language model pipeline was the acquisition of relevant documents. In this case it was the entire documentation of Unity game engine, which was downloaded as HTML-files, split into subfolders and vectorized into a database.

Once the database was usable, a hybrid RAG & LLM system was created to use it as a fact-checker. The system was tested with queries of already solved problems to see if it could guide its user to a matching solution. Tests were conducted with various combinations of RAG and pure LLM models.

Overall, the system managed to reach acceptable solutions, when assisted by the database and RAG and showed significantly improved coherence with it.

Key words: large language model, retrieval-augmented generation

SISÄLLYS

LYHENTEET JA TERMIT

Embedding	LLm rakenteessa oleva välimalli, joka muuttaa tekstin tekoälyn ymmärtävään muotoon ja takaisin.
LLM	Large Language Model. Suuri kielimalli, jota käytetään tekstin generoimisessa.
RAG	Retrieval-Augmented Generation. Vektorihaku, joka on osa koneoppimisen alaluokkaa. Toimii dokumenttien kanssa.
Transformer	Googlen luoma tekoälyarkkitehtuuri, joka matkii ihmisen neuronien toimintaa.

1 JOHDANTO

Kielimallit ovat runsaassa käytössä ohjelmoijien keskuudessa. Useimmat niistä ovat kuitenkin kaupallisia malleja, joita ei voi ajaa paikallisesti. Paikallisia malleja voidaan haluta niiden tuoman yksityisyyden ja sisällönvapauden takia, jollaista niiden kaupalliset kilpailijat eivät voi tehdä sen tuomien pr ongelmien vuoksi.

Yksi suurien kielimallien ongelma on niiden pitkä kehitysprosessi, jonka aikana vanha versio jää ajasta yhä enemmän ja enemmän jälkeen. Tehden siitä epäluotettavan kumppanin nopeasti kehittyvillä aloilla, kuten pelinkehityksessä.

Lähiaikoina LLM tekoälymalleihin on alettu yhdistämään erilaisia moduuleja, jotka hakevat ajantasaisempaa tietoa internetistä, kuvista tai dokumenteista. Tällä pyritään pitämään vanhempi malli relevanttina pitkälle tulevaisuuteen ja tarkentamaan sen tuloksia.

Tämä työ keskittyy paikallisiin tekoälymalleihin, yrittäen löytää keinoja pienten mallien toiminnan tarkentamiseen esimerkiksi dokumenttihaulla ja Retrieval-Augmented Generation tekniikalla. Työn näkökulma tarkastellaan pelinkehityksen linssin kautta sen monialaisen kehitysprosessin takia.

2 KIELIMALLIEN TEORIA

2.1 Mikä on LLM

Suuret kielimallit tai englanniksi Large Language Models (LLM) ovat Tekoälyn alatyyppi, jossa mallien tarkoituksena on tuottaa ja analysoida tekstiä niiden tietokannan mukaisella tavalla. Kielimalleja kutsutaan suuriksi lähinnä niiden laajojen ja monipuolisten tietokantojen takia. Määritelmällä ei ole mitään tarkkaa kokorajaa. LLM tietokantojen kokoluokka on usein kuitenkin kymmenistä gigatavuista aina useisiin teratavuihin asti.

Koulutettua mallia voi tarkemmin kohdistaa “fine tuning” metodilla. Tässä tekniikassa valmis malli koulutetaan osittain uudelleen kohdistetulla datasetillä, jotta mallin osaamista voitaisiin parantaa tietyllä osa-alueella muiden kustannuksella. Esimerkkejä tästä ovat koodaamiseen erikoistetut mallit, joita käytetään tässä opinnäytetyössä.

Nykyiset LLM mallit vaativat toimintaansa, että koko malli ladataan RAM- tai VRAM muistiin. VRAM muisti on tässä käytössä kymmeniä kertoja nopeampaa, mutta sen saanti on rajattua kuluttajille. Nykyiset suorituskykyiset mallit ovat kokoluokkaa 8-200gb tiedostoja ja tämän saaminen muistiin vaatii laitteistolta paljon suorituskykyä. Kuluttajille vastaavat laitteet ovat käytännössä saavuttamattomissa niiden hinnan ja energiankulutuksen takia.

Parametrien koko lisää laitteiston vaatimuksia hyvin nopeasti, varsinkin reaaliaikaisissa sovelluksissa, joissa vastaamisnopeus on äärimmäisen tärkeää sovelluksen käytettävyyden kannalta.

Mallin koko määrittyy useista eri parametreista, jotka jokainen kasvattavat mallin kokoa samassa suhteessa. Ensimmäisenä on itse Transformerit, joiden kokoa voidaan kasvattaa kahdella tavalla. Syvyyden kasvaessa malliin lisätään useampia kerroksia Transformereita kasvattaen sitä ns. korkeussuunnassa. “Leveyden” kasvaessa taas yksittäisiin Transformereihin lisätään huomioneuroneita, jolloin jo olemassa olevien Transformer kerrosten ajattelu- ja suorituskyky kasvavat.

Mallin koko kasvaa myös, jos sen liitännäisiä mekanismeja monimutkaistetaan. Embedding kerrokset hoitavat tiedon muuntamista ja vektorien kulkua mallista sisään ja ulos. Näiden kerrosten kasvattaminen on usein tarpeellista varsinkin, jos malli tarvitsee monimutkaista sanastoa ja lauserakenteita. Kyseisiä rakenteita tarvitaan suurikokoisina usein proosan tai muun pitkän tekstin tuottamisessa. Viimeisenä merkittävänä koon aiheuttajana ovat itse-huomiomekanismit, jotka sisältävät lukemattomia painoarvoa tarkastelevia matriiseja. Nämä ovat merkittävässä osassa Transformerien toimintaa, mutta niiden avaaminen on liian laaja tämän työn kontekstissa.

Parametrien koon kasvattaminen on tärkeätä nykyisissä tensorimalleissa sen tuoman kriittisen hyödyn takia. Oleellisimmat ominaisuudet, jotka paranevat selvästi koon kasvaessa ovat: oppimiskyky, ilmaisukyky ja mallin “tiedon laajuus”.

Oppimiskyky on mallin kyky “sopia” sen koulutettuun tietokantaan. Suurempi malli kykenee sovittamaan tuotoksensa tarkemmin tietokannan sisältöön. Löytäen täten enemmän ja enemmän vivahteita ja yksityiskohtia datasta. Ilmaisukyky taas mittaa mallin kykyä esittää näitä vivahteita ja yleistää tietoaan dataan, jota sillä ei suoraan ole käytettävissä. Ilmaisukykyä mitataan usein erilaisilla loogisilla pulmilla, joissa data kielimallille esitetään sanallisesti hämmentäviä kyselyitä, joissa se usein kompastelee.
Esimerkkinä yleinen sanajärjestystesti:

Arrange the words given below in a meaningful sequence.

1. Police 2. Punishment 3. Crime 4. Judge 5. Judgment

1. 3, 1, 2, 4, 5
2. 1, 2, 4, 3, 5
3. 5, 4, 3, 2, 1
4. 3, 1, 4, 5, 2

Answer: Option D
Explanation: The correct order is :
Crime Police Judge Judgment Punishment

Tekoälyt usein kompastuvat logiikkaan, koska pienikin hallusinointi antaa oikean kuuloisen, mutta loogisesti väärän vastauksen.

Viimeisenä skaalaavana ominaisuutena on ehkä liiankin ilmiselvä mallin tietojen kattavuus. Mitä enemmän mallille syötetään tietoa, sitä enemmän se on tietoinen erilaisista aiheista ja asioista. Varsinkin yleiskäyttöön suunnatuissa tekoälyissä, kuten ChatGPT laajempi tietokanta on tuotteen elinehto.

2.2 Relevantit AI Tekniikat

2.2.1 Transformer arkkitehtuuri

Transformer on Googlen kehittämä tekoälyarkkitehtuuri, joka perustuu monipäiseen huomiomekanismiin. Teksti muunnetaan numerovektoreiksi ja vektorit edelleen embedding taulukon kautta tekoälyn ymmärtäväksi muodoksi. Vanhemmista arkkitehtuureista poiketen Transformer mallit unohtavat rekursiiviset komponentit ja keskittyvät kokonaan huomiomekanismeihin.

2.2.2 Huomiomekanismi

Neurovekkojen ja syävoppimisen näkökulmasta “huomioiminen” on mekanismi, joka mahdollistaa mallin keskittymisen tiettyyn osaan syötedatasta. Tapa, jolla keskittyminen toteutetaan, on kehitetty muistuttamaan ihmisen kognitiivista prosessia, jossa keskitytään valikoivasti tiettyihin osiin aisti-informaatiota ja hylätään muut ärsykkeet. Huomiomekanismi on osoittautunut erinomaiseksi työkaluksi erilaisissa käyttökohteissa, kielen käsittelyssä ja tekoälyn näön kehittämisessä.

Huomiomekanismi juurtaa terminä itsensä varhaisiin koneoppimisen käsitteisiin 1990-luvulta, jollin sitä alettiin johtamaan psykologian ja neurotieteiden vastaavista tutkimuskohteista. Kyseisissä tutkimuksissa selvitettiin ihmisten keskittymistä valikoivasti tiettyyn ärsykkeeseen aisti-informaation käsittelyssä ja miten sitä voitaisiin hyödyntää koneoppimisessa.

2.2.3 MoE

MoE tai Mixture of Experts on tensorimallien alatyyppi, jossa malli sisältää yhden Transformer-arkkitehtuurin sijaan useita “asiantuntijoita”. Jokainen asiantuntija on oma pienempi kielimallinsa, joka erikoistuu tiettyyn osa-alueeseen. Sisään tulevien vektorien rajapinnassa on kielimallin sijaan reititin, jossa embedding malli arvioi mille asiantuntijoille tehtävä annetaan. Yleensä tehtävän suorittaa 1–2 asiantuntijaa, joiden tuotokset yhdistetään ja tulostetaan käyttäjälle.

Moe mallissa asiantuntijoiden “välissä” toimii “Sparse Switch FFN” kerros, jonka tehtävänä on toimia itsenäisesti muista riippumatta ja ohjata kytkimen kanssa syötetty informaatio oikealle asiantuntijalle. MoE mallien Kytkimen kouluttamisessa joudutaan usein lisäämään keinotekoista häviötä, jotta malli keskittyy tasaisesti jokaiseen asiantuntijaan muutaman parhaan suosimisen sijaan.

MoE mallit ovat yleensä hieman pienempiä, mitä parametrien koko antaa olettaa. Esim. 8x7b malli, jossa on kahdeksan 7 miljardin parametrin asiantuntijaa. on 56b sijaan n, 47b, koska FFN kerros on jaettu asiantuntijoiden kesken. Sama tapahtuu mallin käytössä. Samalla tavalla kahden asiantuntijan laskutoimitukset ovat enemmän 12b mallia vastaavat 14b sijaan.

2.2.4 Sub-Agents

Kielimallit voivat myös toimia monimodaalisesti. Monimodaalinen malli käyttää agentteja apunaan suorittamaan erilaisia tehtäviä, joihin tekstimalli ei itse pysty. Yleisimpiä kohteita agenteille voi olla kuvan analysoiminen tai internet lähteen tiivistäminen kielimallille. (Amit Y. 2023)

Kielimallia auttavia agentteja kutsutaan usein “sub-agent” termillä johtaen tavasta, jolla agentit toimivat pääasiassa kielimallin kutsumana. Agentteja voi jakaa erilaisilla luokittelutavoilla, mutta tämän työn puitteissa jako tehdään karkeasti niiden älykkyyden mukaan.

Ensimmäinen ja yksinkertaisin agenttityyppi on SRA tai Simple Reflex Agent. nämä yksinkertaiset agentit reagoivat ainoastaan sen hetkiseen havaintoon, jättäen historian kokonaan huomioimatta. Halutun havainnon täyttyessä agentti suorittaa toimintonsa, muuten mitään ei tapahdu. Kyseinen agenttityyppi on tehokas ainoastaan, kun koko ympäristö on täysin havaittavissa. SRA:ia voi käyttää esim. roskapostin suodattamiseen tai älytermostaattien hallitsemiseen.

Seuraavana älykkyydessä on Model-Based Reflex agenttityyppi, joka toimii hyvin samalla periaatteella, kuin SRA: säännön täyttyessä agentti suorittaa siihen ohjelmoidun toiminnon. Merkittävin ero ensimmäiseen tyyppiin on agentin sisäinen malli, joka kuvaa sen ympäristön funktioita. Sisäinen malli päivittyy aina mallin saadessa uutta informaatiota sensoriensa kautta.

Tämä sisäinen malli mahdollistaa agentin toiminnan ympäristössä, josta se kykenee havainnoimaan vain osan. Mallin rakenne kuvaa näitä piilossa olevia osia ympäristöstä ja sisältää tarpeellisen tiedon niiden kanssa toimimiseen. Agentin toiminta vaatii kahden datapisteen ymmärtämisen. Ensimmäisenä pitää tietää miten ympäristö muuttuu ilman mallin toimintaa ja toiseksi, miten mallin toiminta vuorovaikutaa siihen.

Tämä kaksiosainen ymmärtäminen mahdollistaa mallin tietoon perustuvan toiminnan ympäristössä, jossa kaikki ei ole tiedossa. Esimerkkinä toimii hyvin tekoäly sokkelopelissä. Agentti tietää labyrintin säännöt ja navigoi sitä pitkin, vaikka se ei näe koko sokkeloa tai siellä liikkuvia muita pelaajia.

Tehtävien monimutkaistuessa agenttien sisäinen toiminta muuttuu samaa tahtia. Kolmas ryhmä tiivistetään usein termiin Goal-Based Agents. Nimensä mukaisesti kyseiset agentit on suunniteltu tarkasti pääsemään tiettyyn tavoitteeseen. Erona aikaisempiin agenttityyppeihin on ongelmanratkaisukyky.

Näiden agenttien ohjelmointitapa mahdollistaa tavoitteeseen navigoimisen lukemattomien vaihtoehtojen kautta. Monimutkaisempi rakenne antaa niille työkalut tehdä päätöksiä esim. suunnittelemalla “reittiä” tavoitteeseen tai erinäisten työkalujen käyttö tiedon lisäämiseksi. Näitä agentteja on myös erityisen helppo muokata, jotta ne voivat mukautua aina uudenlaisiin tilanteisiin.

Suositut kaupalliset LLM mallit, kuten ChatGPT ja googlen Bard toimivat osassa tilanteissa tavoitteellisina agentteina. Mallit pyrkivät antamaan parhaan mahdollisen vastauksen käyttäjien kyselyihin, käyttäen kaikkia saatavilla olevia työkaluja. Nämä työkalut saattavat sisältää tietokantojen lisäksi internet hakuja tai kuvien analysointia.

Hyötypohjaiset agentit tai Utility-Based Agents on tyyppi, jonka älykkyyden tasolla käyttöympäristö siirtyy pois rajatusta tilasta. Ne tähtäävät edelleen tiettyyn lopputulemaan tehtävässään, mutta ne on optimoitu tiettyyn käyttökohteeseen, joka voi olla aina sijoitus avustajasta energian optimoijaan.

Edellisistä agenttityypeistä eroten nämä agentit eivät pyri yhteen tiettyyn ratkaisuun, vaan ne pyrkivät edulliseen lopputulokseen asetettujen kriteerien mukaisesti. Hyötypohjaiset agentit ovat parhaimmillaan tilanteissa, joissa halutuissa lopputuloksissa on useita vaihtoehtoja.

Näissä tilanteissa agentit optimoivat työskentelyään päästäkseen parhaimpaan lopputulokseen. Optimointi tapahtuu erilaisten syötettyjen parametrien mukaan ja voi pyrkiä esim. parhaaseen lopputulemaan, nopeasti siedettävään lopputulemaan, tai lopputulemaan, joka täyttää agentin sisäisen arvioinnin parhaiten. Sisäistä arviointia käytetään ottamaan huomioon tilanteiden mahdolliset epävarmuudet, jolloin agentti ei pysty arvioimaan tilanteita konkreettisin mittarein.

Viimeinen ja älykkäin agenttiluokka on Learning Agents tai oppimisagentit. Nimensä mukaisesti agentit oppivat ja säätävät itseään jatkuvasti toiminnan aikana. Oppimisagentit koostuvat pääasiassa neljästä perus konseptikomponentista:

1. Oppimisen elementti: Tämän osa-alueen tehtävänä on parantaa suorituskykyään omaksumalla tietoa vuorovaikutuksesta ympäristön kanssa.
2. Kriitikko: Oppiva elementti saa palautetta kriitikoilta, jotka antavat arvion siitä, miten hyvin agentti suoriutuu verrattuna ennalta määritettyyn suoritusstandardiin verrattuna.
3. Suorituskykyelementti: Vastaa ulkoisten toimien valinnasta perustuen oppimisen kautta hankitun tiedon perusteella.
4. Ongelmanmuodostaja: Tämä moduuli vastaa sellaisten toimien ehdottamisesta, jotka johtavat agenttia tuoreisiin ja kehittäviin kokemuksiin.

Hyvä esimerkki oppimisagentista on AutoGPT. Saadessaan tehtävän AutoGPt tutkii verkkoa ja omaa tietokantaansa, saadakseen kattavan vastauksen kyselyyn. Verkon tutkimisessa se arvioi myös sivujen luotettavuutta oman aliagenttinsa avulla. Tehtävän lopuksi AutoGPT tulostaa kattavan raportin käyttäjälle löydöksistään.

Agentit voivat myös toimia useamman agentin systeemeissä. Näitä kutsutaan yleisesti MAS termillä tai Multi-Agent Systems. Näissä systeemeissä agentit “keskustelevat” toistensa kanssa, tehostaen toimintaansa. MAS:it voivat olla rakenteeltaan, joko homo- tai heterogeenisiä. Homogeenisissä systeemeissä agentit ovat samanlaisia ja heterogeenisissä vaihtelevan kykyisiä toimintansa osalta.

Erilaisuus on yleisesti ns. kaksiteräinen miekka, jossa heterogeeninen systeemi tekee sisäisestä kommunikoinnista monimutkaista. Tämä monimutkaisuus taas voi olla, joko puhdas hidaste, tai se voi tehdä systeemistä hyvin mukautuvan ja joustavan.

Riippumatta agenttien tyypeistä, ne voidaan myös asettaa, joko kilpailemaan toistensa kanssa tai toimimaan yhdessä tavoitetta kohti. Yksittäinen MAS voi sisältää molempia toimintatyyppejä samanaikaisesti sen tarpeen mukaisesti. Agentit voidaan myös asettaa jäykkään hierarkiaan, jolloin yleensä oppimisagentti käyttää yksinkertaisempia agentteja apunaan esim. web haussa, kuvien analysoinnissa tai RAG tehtävissä.

2.2.5 RAG

Retrieval-augmented generation tai lyhennettynä RAG on dokumenttien hakemiseen tarkoitettu koneoppimisen alle luokiteltu systeemi. Tekoälymallien yleinen haaste
on niiden tietokantojen ajantasaisuus ja laajuus. LLM mallit, joita halutaan käyttää tiettyihin yksittäisiin tarkoituksiin, pitäisi kouluttaa vain tämän aihealueen tiedoilla. Tämä tietenkin rajoittaisi mallin toiminnan vain tähän alueeseen ja kouluttamisen kalleuden takia tätä ei usein voida tehdä.

Tähän ongelmaan tuli ehdotus ratkaisuksi Metalta vuonna 2020. Meta julkaisi blogin (Meta RAG blog, 3), jossa se avasi RAG-metodin toimintaa ja käyttötarkoituksia. Tekoälymalli, joka voi tutkia ja kontekstualisoida tietoa on tällä hetkellä vielä tavoittamattomissa. Tähän Meta ehdottaa RAG-metodia, jossa malli saa avukseen vektoritietokannan, josta RAG hakee kyselyä vastaavia dokumentteja ja asettaa ne mallin saataville, jolloin malli voi verrata tuloksiaan tarkkoihin dokumentteihin ja korjata vastauksiaan.

RAG toimii kuten perinteinen sequence2sequence malli, joka ottaa yhden sekvenssin sisään ja tulostaa toisen vastaavan. Ero perustaviin malleihin on kuitenkin RAG:n kyky käyttää syötettyä kyselyä relevanttien dokumenttien hakemiseen.

Esimerkkinä kysymys: “Mikä oli historian suurin rakennus?” Vektorien samankaltaisuushaku saattaa nostaa esille asiakirjoja aiheista “Suuret rakennukset”, “Rakennusten historia”, ja “Kuuluisat arkkitehdit”. Nämä asiakirjat syötetään RAG-malliin, joka tuottaa lopullisen tuloksen. RAG:lla on siis kaksi tietolähdettä. Mallien parametreihin tallennettu tieto (parametrinen muisti) ja sen “aivoissa” oleva tieto (parametriton muisti).

Nämä kaksi muistilähdettä täydentävät toisiaan ajattelun osina. Testeissä on myös havaittu RAG:n käyttävän parametrista muistiaan toisen “vihjeinä”, jotta se tuottaisi oikeita vastauksia kyselyihin. Tällä metodilla voidaan yhdistää "suljetun kirjan" tai vain parametreihin perustuvan lähestymistavan joustavuus "avoimen kirjan" tai hakupohjaisten menetelmien suorituskykyyn.

RAG käyttää myös eräänlaista fuusioyhdistämistä haettujen asiakirjojen tiedon integraatioon, mikä tarkoittaa yksittäisten vastausennusteiden yhdistämistä asiakirjojen sisältöön ja yhdistää näiden lopulliset ennustepisteet. Suorituskykyä parantaakseen virheilmoitukset syötetään takaisin hakumekanismiin mikä nopeuttaa koko prosessia huomattavasti.

3 JÄRJESTELMÄN SISÄLLÄ

3.1 Yksityisyys

Verkossa toimivat kaupalliset tekoälyt ovat saavuttaneet maailmanlaajuisen suosion ja niiden käyttäjäkunta on läpäissyt kaupalliset yritykset läpikotaisin. Tämä suosio on kuitenkin tuonut yritykset uuden ongelman äärelle. Suuret kielimallit koulutetaan käyttäjien tuottamalla datalla ja mikä sen parempaa dataa, kuin alojen ammattilaisten tekemät kyselyt. Yrityksille datan pääseminen ulos turvallisesta ympäristöstä on kestämätöntä ja saatava estettyä keinolla millä hyvänsä. Generoivien tekoälyjen käyttöä ei kuitenkaan voi lopettaa niiden tuomien tuottavuushyötyjen takia. (Reuters AI privacy paradox)

Tämä jättää toimijat kahden vaihtoehdon äärelle, joko kaikki suoritetaan paikallisesti omalla laitteistolla, tai turvaudutaan uusiin “enterprise” luokan paketteihin, jossa tekoäly-yritykset tarjoavat samaa toimivaa kielimalliaan, mutta tällä kertaa datan luvataan pysyvän yrityksen hallinnassa, eikä sitä käytettäisi mallin uudelleen kouluttamiseen.

Yritykset ovat kuitenkin jääneet toistuvasti kiinni tekoälyn kouluttamisesta laittomasti tai kertomatta kerätyn tiedon kanssa. Asiaa kärjistää vielä se, että valtioiden lait eivät ole ajantasaisia ja täten eivät pysty estämään yritysten toimintaa hypermodernilla alalla. (Politico AI ruling). On myös esitetty syytöksiä suosittujen mallien tarkoituksellisesta suunnitelusta yksityisyyttä huomioimatta. (Techcrunch Chatgpt privacy design).

Paikallinen tapa ajaa malli on usein kallista ja suurimpien kohdalla hyvin epäkäytännöllistä. Laitteisto on hyvin kallista ja kuluttaa huomattavan paljon virtaa. Pienet mallit taas eivät ole tarpeeksi “älykkäitä” työnteon vaatimaan tarkkuuteen nähden. Opinnäytetyössä onkin tarkoituksena selvittää pienten ja hyvin erikoistuneiden mallien käyttöä RAG-avusteisena tämän ongelman korjaamiseksi.

3.2 Hallusinointi ja luotettavuus

(Amaratunga 2023, 5.) Tekoälyt kärsivät usein ilmiöstä nimeltä hallusinointi. Hallusinoidessaan tekstimalli generoi epätodellista informaatiota. Käytännössä malli alkaa kehittämään todelliselta kuulostavia “faktoja” tyhjästä. Pahimmillaan tämä johtaa hyvin luotettavan tuntuiseen valehtelevaan malliin, joka ripottelee aidon tuntuisia ”faktoja” oikean tiedon sekaan. Syitä hallusinointiin voi olla useita ja niiden vaikutukset voivat myös sekoittua arvaamattomilla tavoilla. Hallusinointien yleisimpiä lähteitä on kolme: koulutusdata, kysel ja datan vanhentuminen.

Ongelmat koulutusdatassa voivat johtaa hallusinointiin useilla eri tavoilla: Koulutusdatasta yleistämisessä malli yleistää miljardeista datapisteistään asioita liian pitkälle ja tulostettu teksti menee asiasta ohi ja muuttuu virheelliseksi.
“Pohjatotuuden” puuttuessa tekoälyllä ei ole suoraa vastausta koulutusdatassaan. Kyseinen virhe ei ole niin suuri ongelma esim. kuvia luokittelevassa mallissa, joka tietää oikean kuvan ulkonäön ja näin vastauksen kysymykseen. LLM:ät on kuitenkin suunniteltu tuottamaan tekstiä lukemattomiin eri tarkoituksiin ja täten oikeaa vastausta ei ole. Tämän seurauksena mallien on usein vaikeata tuottaa “oikea” vastaus.

Koulutusdata voi myös olla itsessään epätäydellistä. Ylisovitettu data ajaa mallia toistuvasti samanlaisiin lopputuloksiin kyselyistä riippumatta, pilaten vastausten laadun. Koulutusdata voi myös olla puolueellista, joka on käytännössä tietoista ylisovittamista ja johtaa samanlaiseen lopputulokseen. Kolmas ongelma syntyy, kun LLM alkaa toistamaan tiettyjä kuvioita, lauseita tai tiedonpalasia harjoitusdatasta, vaikka ne eivät olisikaan merkityksellisiä käyttäjän kannalta.

Generoinnit voivat myös saada häiriötekijöitä ulkopuolelta. Käyttäjän kysely vaikuttaa omalta osaltaan generointiin, ja maallikkojen kyselyt ovat harvoin optimoituja tekoälyn luettavaksi. Käyttäjien kirjoittama kysely voi pelkän muotoilunsa takia muuttaa generoitua vastausta huomattavasti ja epäselvät tai johtavat kyselyt usein vääristävät generoitua vastausta huomattavasti.

Viimeinen yleisistä ongelmista syntyy seurauksena tekoälyjen faktantarkistuksen puutteesta. Kielimallit eivät virallisesti vastaa kysymyksiin, vaan generoivat todennäköisimmän vastauksen tekstin vektorisoituun muotoon perustuen. Mallien vastaukset ovatkin usein hyvin todellisen näköisiä, riippumatta niiden varsinaisesta todenmukaisuudesta.

Hallusinointiin puuttumiseksi mallien kehittäjät pyrkivät hienosäätämään malleja tarkemmilla ja yksityiskohtaisilla tietokannoilla. Lisäten ihmisen koulutukseen mukaan valvomaan sen suuntaa. Kaupalliset toimijat ovat myös lisänneet ulkoisia tarkistuksia käyttäjän ja mallin väliin pyrkiessään välttymään skandaaleilta mallin generoimien tekstien takia.

3.2.1 Mallien kokorajat ja niiden vaikutukset

Eri käyttötarkoituksiin tarkoitetut tekoälymallit ovat pohjimmaiselta rakenteeltaan hyvin samanlaisia. Rakenteet pohjautuvat valtaosin tensoreihin, joiden läpi syötetty informaatio kulkee matemaattisten mallien ohjaamana ja ulos tulostetaan vastaus. Oli se sitten kuva, ääni tai teksti. Samanlaisesta rakenteesta huolimatta eri tarkoitukseen tehdyt mallit reagoivat hyvin omalaatuisilla tavoilla kyseisen rakenteen muutoksiin.

Mallien koot ilmaistaan usein miljardeissa parametreissa nimen jälkeen esim. “Malli 3b”, jolloin malli on n, kolmen miljardin tensorin (Amaratunga, T. 2023) kokoinen. Sujuvan ja järkevän toiminnan vaatima koko on kuitenkin hyvin riippuvaista käyttötarkoituksesta.

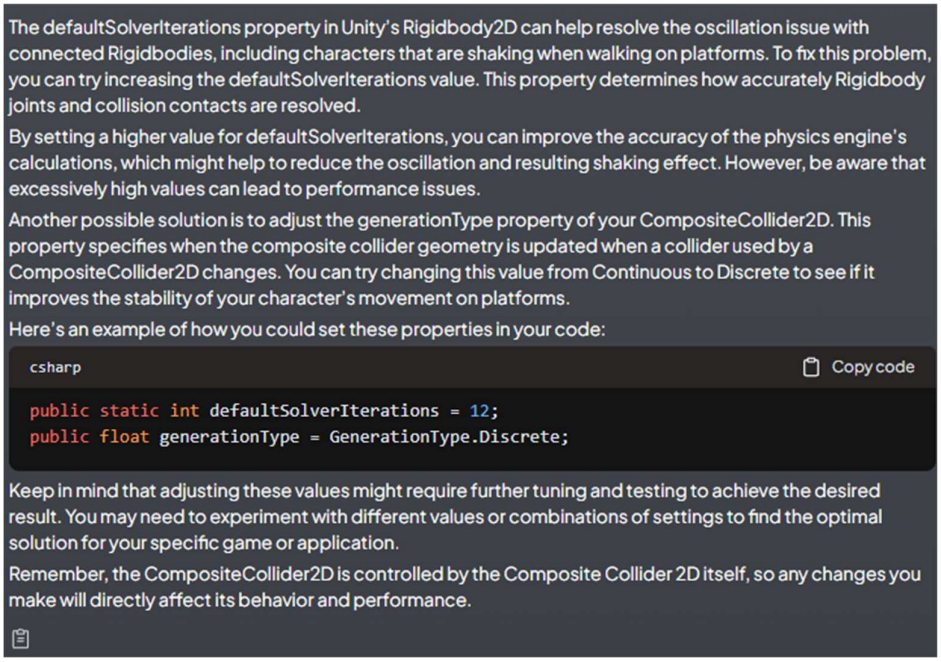
Avoimen kuvangeneroinnin kärkinimi Stable diffusion XL malli sisältää version mukaan 3-6mrd parametria ja kykenee osaavan käyttäjän käsissä äärimmäisen vaikuttaviin tuloksiin. Tekstin generaatiossa taas 7mrd parametrin mallit ovat usein minimi edes jotenkin järkevään käyttöön ja ne kadottavatkin usein kontekstin, sekoittavat henkilöitä keskenään tai kohtaavat lukemattomia muita ongelmia tekstiä generoidessaan. Tekstingeneroinnissa suurimmat vapaasti käytettävät mallit ovat yli 120 miljardia parametria suuria kasvaen 20-kertaisiksi kuvangenerointiin verrattuna.

Kokoerojen ongelmat tulevat hyvin ilmi, kun mallien tehokkain ajamisympäristö on näytönohjaimien VRAM, joka on tavalliseen RAM:iin paljon rajoitetumpi resurssi. Kuluttajakortit, joilla malleja on järkevää ajaa sisältävät n,8–24 gigatavua VRAM:ia, joka riittää pääasiassa kuvangenerointiin omalla laitteistolla. Tekstingeneroinnissa edes suurimman hintaluokan 24gt kortit riittävät juuri ja juuri alimman kolmanneksen kokoluokkiin, joissa mallien koot liikkuvat 13-24mrd parametrissa. Suurimpien tekstimallien ajamiseen tarvitaan, joko useita parhaita näytönohjaimia samanaikaisesti tai kymmeniä tuhansia euroja maksavia ammattilaiskäyttöön tarkoitettuja kiihdytinkortteja. Tämä tekee tekstingeneroinnista verrattaen vaikeasti saavutettavaa tavalliselle kuluttajalle verrattuna muihin tekoälyn aloihin.

3.2.2 Vektoritietokannat

Vektoritietokanta on kokoelma numeroilla esitettyä dataa. Data on yleensä muodossa {1, 2, 3, …}. Tekoälyille tämä vektoreina ilmaistu data on tällä hetkellä tehokkain tapa saada maksimaalinen määrä informaatiota käyttöön. Tärkein etu perinteiseen tietokantaan on vektorien mahdollistama tietojen samanlaisuuden vertaaminen perinteisten tarkkojen vastaavuuksien hakemisen sijaan. Tietokantoja käytetään myös yleisesti koneoppimisessa, mutta työ keskittyy LLm malleihin (Cloudflare Vector database).

Alla olevassa kuvassa haetaan vektoritietokannasta vastaus kysymykseen ”Composite collider causes character to shake when walking on platform. how to fix?” Vastaus ei käytä tekoälyä, vaan hakee kyselyn sisältämän tekstin perusteella ”samankaltaisia” vektoreita tietokannasta. Käyttäen 2.2.5 mainittua metodia.



Kuva 1 Samanlaisuushaku Vektoritietokannasta

Samanlaisuuden kautta hakeminen on mahdollista vektorien “klusteroinnin” takia. klusteroinnissa samankaltaiset ja todennäköisesti relevantit vektorit ryhmitellään yhteen tietokannassa. Vektoriklusterien kautta LLM voi yhdistää konteksteja ja sanoja toisiinsa, mahdollistaen ihmiskielen ymmärtämisen ja järkevän tekstin generoimisen.

Modernit chat käyttöliittymät pystyvät käyttämään vektoritietokantoja pitkäaikaisena muistina. Ohjelmistot vektorisoivat viestejä, jotka ovat jäämässä tekoälyn konteksti-ikkunan ulkopuolelle. Vektorisoituja viestejä lisätään tämän jälkeen samanlaisuus haun avulla kontekstiin. Auttaen tekoälyä muistamaan asioita kauempaa chat historiasta (Vector context Sillytavern documentation).

3.3 Datan ajantasaisuus

Yksi suurimmista ongelmista, mitä tekoäly kohtaa on datan ajantasaisuuden ylläpitäminen. LLM mallit juontavat datansa tensoritietokannasta, jonka kokoaminen ja kouluttaminen kestää kuukausista vuosiin. Vaadittu datamäärä on myös niin suurta, että pelkästään tietokannan kokoaminen kestää pitkiä aikoja.

Tämän seurauksena tekoälyjen tietokannat ovat auttamatta aina vanhentuneita jo valmistuessaan. Tekoälyn osa-alueelle, jonka pääasiallinen tarkoitus on vastata käyttäjien kyselyihin ja pyyntöihin, vanhentunut ja virheellinen tieto on viimeinen asia, jota niihin halutaan. Suurien mallien kehitysvauhti on kuitenkin rajoittunutta ja datan ajantasaisena pitäminen täytyy tehdä kiertoteiden kautta.

Yksi näistä tavoista on kappaleessa 4 avattu RAG tai Retrieval-augmented generaation. RAG vektorisoi dokumentteja ja täten mahdollistaa LLM:n tietojen aukkojen täyttämisen. RAG vaatii tiedostojen vektorisoinnin ja tämän takia se sopii parhaiten rajatun aihealueen tiedon varastoimiseen. Monialaisten kaupallisten tekoälyjen käytössä RAG-dokumenttien tarvittava määrä kasvaisi niin valtavaksi, että sen käyttö on mahdotonta nykyisellä teknologialla.

LLM kehittäjät ovat kehittäneet kokeellisia keinoja, joilla mallit voivat hakea verkosta päivitettyä tietoa käyttäjän kyselyn perusteella. Kyseiset teknologiat kuitenkin kärsivät verkkosivujen erilaisten rakenteiden ja hakukoneoptimoinnin aiheuttamista ongelmista. Nämä ongelmat tuottavat usein vääränlaisia ja epäsopivia verkkosivuja tekoälyn hakutuloksiin. Heikentäen haun hyötyjä merkittävästi.

mallien informaatiota voi myös päivittää, mutta tämän hinta kasvaa massiivisesti mallin koon mukana, koska koulutus pitää tehdä uudelleen aina tietokannan päivityksen jälkeen. Tämä vaatii usein satoja tai tuhansia ammattilaistason näytönohjaimia.

4 KÄYTTÖÖNOTTO

4.1 Pohjustus

Rag-systeemi (2.2.5) valittiin opinnäytetyöhön tekoälyjen yleisen ongelman takia, josta mainittiin kappaleessa 3.3. LLM mallin tekeminen on kallis ja laaja työurakka ja mallin kouluttaminen myös kestää kuukausista vuosiin. Tämä viive tekee niistä verrattaen heikompia työkaluja pelien kehittämisessä, alan äärimmäisen kehitysvauhdin takia. Varsinkin kaupalliset pelimoottorit voivat tehdä suuriakin muutoksia vuoden ajan sisällä, jolloin edes uudehkot LLm:ät eivät sisällä tähän tarvittavia tietoja.

Työn RAG osuuden tavoitteena on selvittää, että voiko Retrieval augment generaation:ia käyttää LLM tekstimallin tukena, dokumentaation ajantasaistamiseen. tekstimallille annetaan pääsy vektoritietokantaan, josta se voi teoriassa noutaa ajantasaiset esimerkit ja ohjeet skriptien käyttöön. Tämä yhdistettynä sen sisäiseen kykyyn ohjeistaa koodin kanssa

RAG-kokeen dokumenteiksi valittiin koko Unity pelimoottorin tietokanta, josta oli saatavilla ladattava html-tiedostoja sisältävä zip kansio. Paketissa oli myös, mukana koko rakenne, jolla verkkosivuston voi käynnistää paikallisessa ympäristössä. Nämä poistettiin ja lopulta jäljelle jäi ~30 000 html tiedostoa, joista yli 90 % kuuluivat scriptreference luokkaan. Loput tiedostot sisältyivät erilaisiin Unityn manuaaleihin.

Scriptreference luokka sisältää yksittäisten unity scriptien tietoja, — ja koodiesimerkin sen käytöstä. Manual- luokka keskittyy itse moottorin käyttöohjeisiin ja sen kanssa toimimiseen koodin kautta.

Tehty RAG-systeemi koostuu neljästä osasta, joista kolme ajetaan Docker-virtualisointikonteissa ja yksi natiivina. Docker mahdollistaa ohjelmiston ja sen ym-

päristön ajamisen omassa virtuaalisessa kontissaan WSL2:n avulla. Täten voidaan ajaa hyvinkin monimutkaisia ympäristöjä helposti ja tehokkaasti myös Windowsissa.

RAG - Ai yhdistelmä suoritetaan kokonaan paikallisessa ympäristössä. Tämän syynä on yritysten tarpeet salassapidosta ja yksityisyydestä. Erityisen tärkeää tämä on, koska tekoäly-yritykset käyttävät avoimesti käyttäjiensä dataa hyväkseen tuotteensa kehittämisessä. Toinen syy paikalliseen ympäristöön on sensuroinnin ja kaupallisten tekoälyjen käyttäjille näkymättömien sääntöjen kiertäminen. Koodin osalta tämä ei välttämättä ole relevanttia, mutta yleisesti tiedostohaussa ja tekoälyn kanssa keskustelun osalta näkymättömien vastausta muokkaavien “vipujen” välttäminen on tärkeää.

Työn aluksi ladattiin Unityn 30k+ tiedoston tietokanta, joka piti pilkkoa järkeviin osiin, koska käyttöliittymän havaittiin romahtavan kokonaan, kun yksittäisen erän koko kasvoi yli tuhanteen tiedostoon. Tiedostojen erotteleminen omiin kansioihinsa tehtiin seuraavalla .bat skriptillä.

```
@Echo Off
If /I Not "%__CD__%"=="%~dp0" PushD "%~dp0" 2>Nul||Exit/B
taskkill /f /im explorer.exe >nul
taskkill /f /im SearchIndexer.exe >nul
sc stop WSearch >nul
sc config WSearch start= disabled >nul

SetLocal EnableDelayedExpansion
Set "DirN=-1"

:Check_DirN
Set/A "DirN+=1"
If Exist "%DirN%" GoTo Check_DirN
cls
echo Moving files to Directory %DirN%...
Set "limit=500"
MD "%DirN%"
For %%A In (*.html) Do (
    RoboCopy . "%DirN%" "%%A" /MOV 1>NUL
    Set/A "limit-=1"
    If !limit! Lss 0 GoTo Check_DirN
)
Echo(Task Done!

start explorer.exe
start SearchIndexer.exe
sc config WSearch start= delayed-auto >nul
sc start WSearch >nul
Timeout -1 1>Nul
```

Kuva 2 Tiedostojen erotteluun tarkoitettu skripti

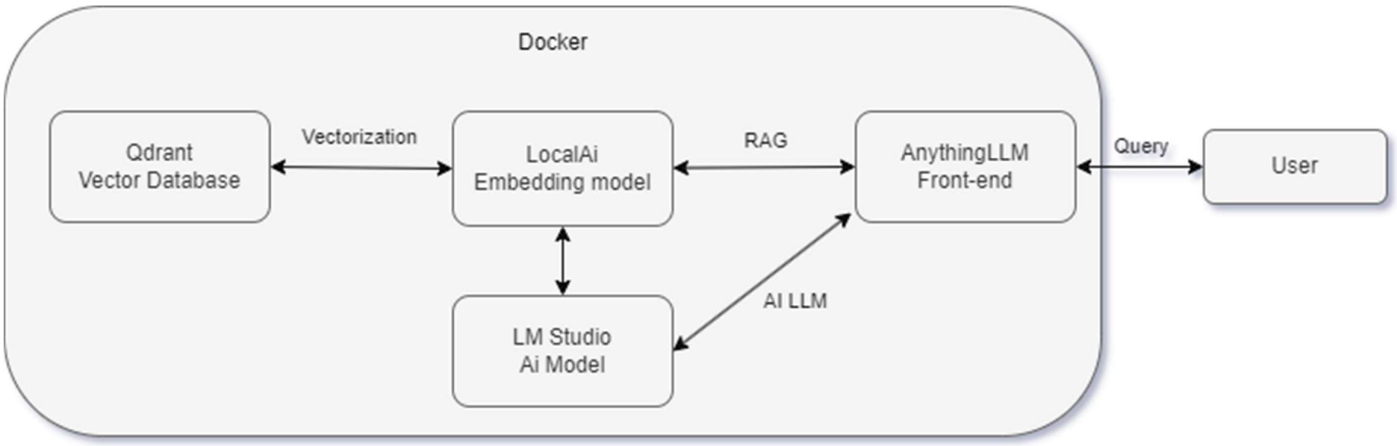
Skripti erottelee tiedostot numeroituihin kansioihin 500 tiedostoa per kansio ja pyrkii pitämään searchindexer.exe:n, sekä explorer.exe:n pois päältä näiden aiheuttaman mahdollisen hidastelun takia.

Alkuvaiheessa vektorisoinnissa jouduttiin käyttämään pelkästään CPU:ta ja tämä oli huomattavan hidasta (n, 35min per 500 tiedostoa). Vektorisointi oli myös taipuvainen jäätymään kokonaan kohdissa, joissa tiedostoja listattiin suoraan levyltä. Puolen välin jälkeen näytönohjain saatiin käyttöön ja vektorisointiin kuluva aika väheni n, viiteen minuuttiin per 500 tiedostoa.

Valittu käyttöliittymä saattaa olla liian kevyeen työskentelyyn tarkoitettu. Tiedosto-listan hakeminen itsessään saattaa kaataa koko Docker-kontin ja vaatia sen uudelleen käynnistämistä. Kyselyjen ajaminen sen sijaan ei aiheuta ongelmia ja on nopea, sekä vakaa.

4.2 Työn osuus

Tehty RAG-systeemi koostuu neljästä osasta, jotka ajetaan Docker-virtualisointikonteissa. Docker mahdollistaa ohjelmiston ja sen ympäristön ajamisen omassa virtuaalisessa kontissaan WSL2:n avulla. Täten voidaan ajaa hyvinkin monimutkaisia ympäristöjä helposti ja tehokkaasti myös Windowsissa.



Kuva 3 Rag-LLM kaavio

4.2.1 Qdrant vektoritietokanta

Qdrant on vektoritietokanta ja sisältää vektorien samanlaisuuksiin perustuvan hakutoimminon. Qdrant valittiin työhön sen mainostetun tehokkuuden ja paikallisen Docker vaihtoehdon vuoksi. Työssä RAG-ominaisuutta varten vektorisoidut HTML-tiedostot tallennetaan tähän tietokantaan tekoälyn käytettäviksi vektoreiksi.

Vektori-indeksejä käytetään yleisesti hakutuloksia nopeuttamassa esim. FAISS (Facebook Research FAISS). Nämä indeksit voivat nopeuttaa hakutuloksia huo-

mattavasti, mutta niistä puuttuvat tarvittavat tietokantojen ominaisuudet. Vektoritietokannat taas sisältävät indeksejä, joita tietokannalla voidaan käyttää erilaisiin tarkoituksiin, kuten työn RAG-haku.

4.2.2 Localai, embedding & LM Studio LLM

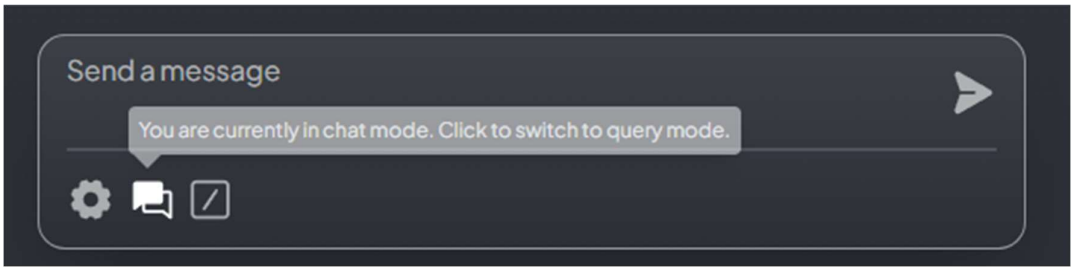
LocalAi ja LM Studio ovat molemmat Llamacp malleja ajavia ohjelmia. Ohjelmia tarvitaan kaksi, koska ajettavat embedding ja LLM AI mallit tarvitaan samanaikaisesti. LocalAI ajaa vektori embedding mallin Dockerin sisällä, koska kyseinen malli vaatii WSL2 ympäristön. LM Studio voidaan ajaa Windows natiivina ohjelmana, välttäen virtualisoinnin vaatimat lisäresurssit.

4.2.3 Anythingllm front-end

Anythingllm valittiin työn käyttöliittymäksi sen ominaisuuksien soveltuvuuden takia. Ohjelma tukee RAG vektori tietokantoja, sekä niiden yhdistämistä LLM tekoälyn tietoihin, sisältäen täten kaiken työhön tarvittavan. Kaikki mallit voi lisäksi ajaa paikallisesti, välttäen kaupallisten tekoälyjen “black box” ongelmat, jossa niistä ei näe muuta, kuin lopputuloksen.

Ohjelmistossa tehdään erillinen työtila jokaiselle tietokannalle, jolloin ne pysyvät erillisinä, eivätkä eriaiheiset tietokannat “saastuta” vastauksia aiheeseen liittymättömällä informaatiolla.

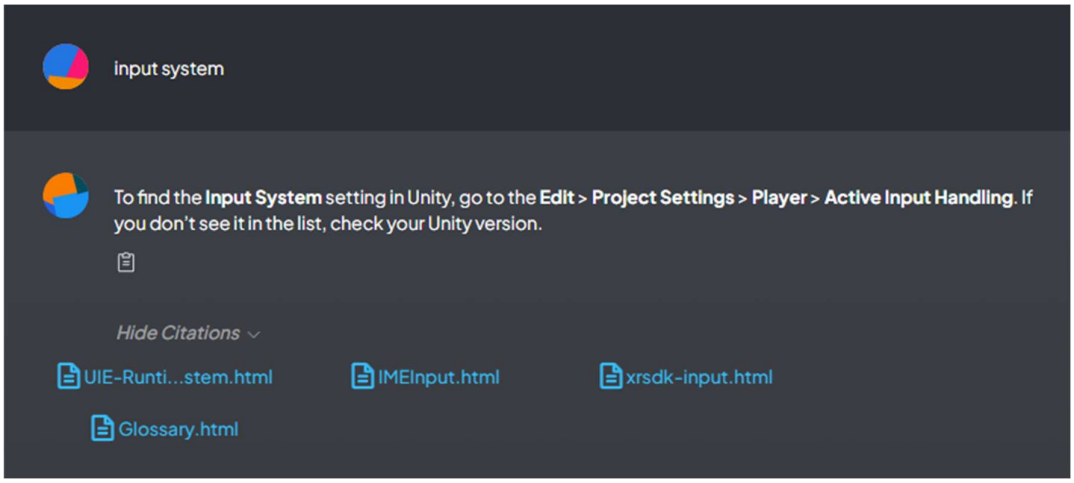
Ohjelman minimalistisessa chat ikkunassa, voidaan valita, joko suora dokumentaatio kysely tai RAG avustetun tekoälyn kanssa keskustelu. Tilan voi vaihtaa, myös kesken viestin kirjoittamisen.



Kuva 4 AnythingLLM chat vaihtoehdot

4.2.4 Document query

“Document query” jättää LLM mallin pois välistä ja kommunikoi suoraan embedding mallin kanssa. Tämä tila on ns. puhdas RAG, jossa tekoäly vastaa ainoastaan asioilla, joita se löytää vektoritietokantaan tallennetuista dokumenteista.



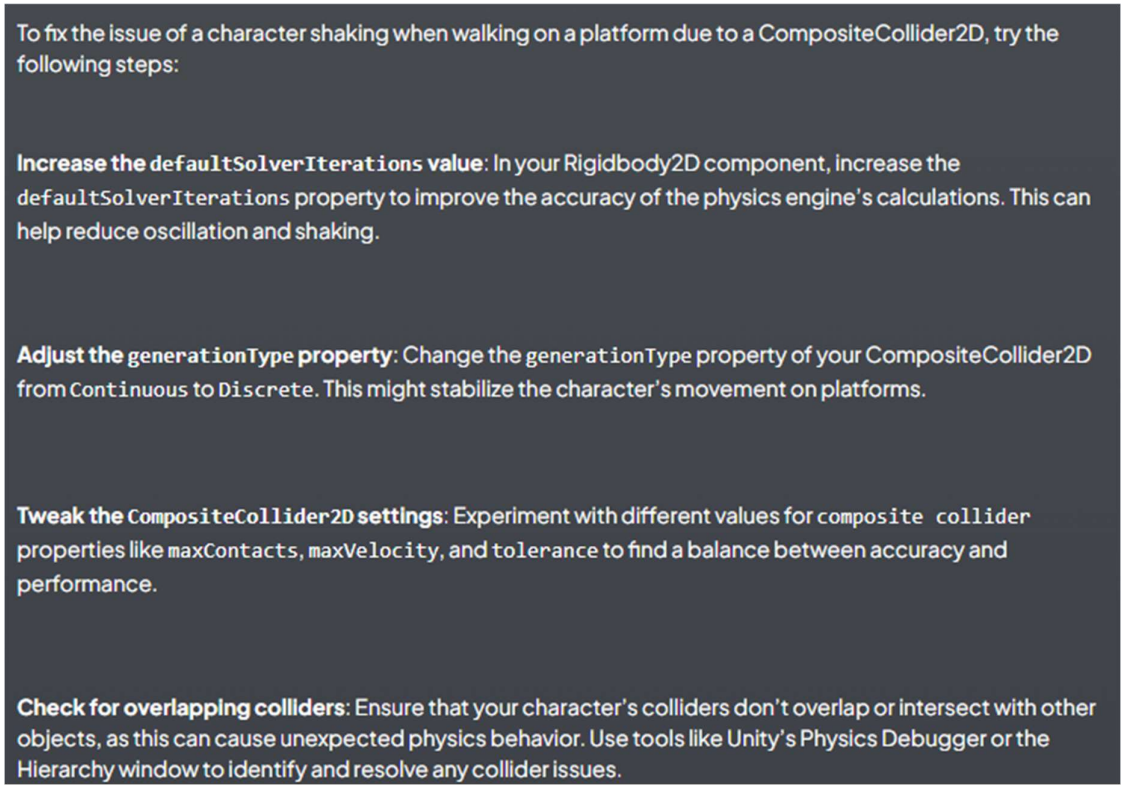
Kuva 5 Document query suoraan Vektoritietokannasta

Opinnäytetyön yhteydessä tietokanta on täytetty pelkästään Unity:n dokumentaatiolla, varmistaen sen aiheenmukaisuuden. Lopputuloksena tämän tilan parhaiten toimiva käyttötarkoitus on toimia äärimmäisen nopeana (~5sekunttia per haku) dokumentaation hakutyökaluna.

Embedding mallin kyky yhdistää hyvinkin yksinkertaiset hakutermit ja kontekstit toisiinsa tekee kyseisestä mallista tehokkaan työkalun dokumentaation hakuun. Referenssi hakukoneena toimiessaan RAG välttää myös tekoälyjen yleisen hallusinointi (3.2) ongelman, koska kaikki palautettu teksti on suoraan dokumenteista.

4.2.5 AI-RAG

“Chat mode” yhdistää RAG hakutoiminnot AI mallin kanssa, joka käyttää vektoritietokantaa ns. toisina aivoinaan käyttäen niihin tallennettua dataa vaihtoehtoisena tietolähteenä. Toimiessaan oikein tämä mahdollistaa tekoälyn tietojen ajantasaisena pitämisen. Opinnäytetyön viitteissä tämä tarkoittaisi nopeasti kehittyvän Unityn kehityksen mukana pysymistä, parantaen hakutuloksia merkittävästi.



Kuva 6 Sama kysymys kuvan 1 kanssa (Composite collider causes character to shake when walking on platform. how to fix?), mutta tällä kertaa AI LLM on mukana vastaamassa käyttäen RAG:ta apunaan.

AI LLM:ien tilastollisesti merkittävä testaaminen on käytännössä mahdotonta niiden "black box" olemuksen takia. Tekoälyt toimivat matemaattisten mallien (kappale 2.2.1) kautta ja edes niiden tekijät eivät täysin tiedä mikä saa tekstin tulostumaan tietyllä tavalla.

Tämän takia mallin testaaminen deterministisesti ei ole järkevää, eikä lopputuloksista voi muodostaa täysin objektiivista päätelmää. Huomiomekanismi itsessään auttaa hieman, mutta mallin kokonaisvaltainen ymmärtäminen on kirjoitushetkellä hyvin vaikeaa. (Amartunga 2023, 44).

4.3 Tulokset

4.3.1 Mallit

Mallien vaikutukset tuloksiin tulivat esille työn aikana useampaan kertaan. Kappaleessa 3.2 käsitelty hienosäätö tuottaa samasta pohjamallista hyvin erilaisesti toimiva johdettuja malleja. Selvimmin tämä tulee esiin tekoälyn kanssa keskustelussa, jossa eri alamallit vastaavat omilla lauserakenteillaan ja saattavat jopa

erota osaamistasossaan merkittävästi, vaikka ne on johdettu samasta lähtöpisteestä.

Myös embedding mallit näyttivät eronsa työn aikana. Työn alussa käytettiin AnythingLLM:n sisäänrakennettua embedding mallia, joka hoiti työnsä kelvollisesti, mutta hitaasti. Myöhemmin sain yhdistettyä Dockerin näytönohjaimen rajapintaan ja otin käyttöön huomattavasti tehokkaamman LocalAi:n embedding mallin. Tämä malli toimi yli 50-kertaisella nopeudella, mutta myös tuotti parempia tuloksia, sekä puhtaaseen RAG hakuun, että tekoälyn apuna toimivaan dokumenttihakuun.

Embedding mallien erojen ongelma on niiden vaikea todennettavuus. RAG haku ei ole deterministinen ja muuttaa vastaustaan, jos samaa hakua toistetaan tarpeeksi pitkään.

4.3.2 Kielimallit

Yleisessä tekstingeneroinnissa mallin koko on kohtalaisen verrannollinen sen tuottamaan tekstin laatuun. Suurempi malli tuottaa parempaa ja monipuolisempaa tekstiä. Hienosäädetyt mallit usein toimivat kokoaan paremmin omalla pienellä alueellaan.

Käytetty TinyDolphin-2.8-1.1b malli on kielimallien luokassa mikroskooppinen kooltaan ja normaalissa käytössä se olisi auttamattoman kapea osaamiseltaan ja kirjoitustaidoltaan.

Ohjelmoinnin apuna se kuitenkin ajaa asiansa käytännössä virheettä. RAG:n apu korjasi monessa tapauksessa väärän tai olemattoman kapean vastauksen ja toimi varteenotettavana vaihtoehtona kaupallisille palveluille.

4.3.3 RAG

RAG hakua hypetetään aina välillä uutena suurena asiana, joka korjaa kaikki ongelmat ja tuo geneerisen tekoälyn kaikille. Loppujen lopuksi se on kuitenkin hyvin ”tyhmä” ja sen laatu riippuu massiivisesti sisään syötettyjen vektoridokumenttien laadusta (joiden laatu riippuu embedding mallista jne.)

Jälleen kerran ohjelmointi käyttötarkoituksena auttaa RAG:n tehokkuudessa, koska koodissa riittää vastaavan tekstin palautus dokumenteista. Normaalin tekstin vaatiman kontekstin ymmärtämisen sijaan (2.2.5). Tämä kontekstin sivuttaminen saa RAG:n näyttämään paljon paremmalta, kuin mitä se oikeasti on.

Ohjelmoinnissa riittää erinomaisesti, että se tulostaa tyhmästi kaiken, mikä liittyy vähänkään asiaan. Normaalissa tekstissä se ei kuitenkaan ole omien kokemuksieni mukaan tarpeeksi älykäs yhdistämään kontekstia järkevästi ja johdonmukaisesti.

4.3.4 Yhdistettynä

RAG + LLM on selvä parannus kummankaan yksittäisen systeemin toiminnasta, mutta sen käyttö on myös vaivalloisempaa. Vektorisoidut dokumentit on pidettävä ajan tasalla ja niiden laatua pitää tarkastella vaivalloisilla tavoilla. Tämä tarkoittaisi esim. Unityn tapauksessa koko tietokannan uusimista jokaisen päivityksen kohdalla. Lisäksi verkosta ladattujen sivujen tai dokumenttien laatu luultavasti paranisi, jos niistä siivottaisiin kaikki ylimääräinen pois, kuten HTML- tai CSS muotoilu.

5 POHDINTA

Pelinkehitys on itsessään hyvin vaikea aihe tekoälyille, koska se yhdistää median ja ihmiset ohjelmointiin. Luoden täysin subjektiivisia asioita koodin vaatimuksiksi, joita ns. sanalaskin tekoäly ei voi mitenkään arvioida. Näitä mahdottomia kysymyksiä ovat mm. ”Mikä on hauska peli?” tai ”Mikä tuntuu hyvältä pelata?” Vastaavat kysymykset on jokaisen pelinkehittäjän ratkaistava, eivätkä nykyisenkaltaiset tekoälyt pysty soveltamaan tietoaan sen ratkaisemiseksi.

Asiaa ei mitenkään auta se, että tuloksien tarkastelu saa ihmisen tuntemaan itsensä typerältä. Mallit ovat mustia laatikoita, joten et näe sisälle. Tulokset muuttuvat mallin parametrien mukana ja eivät ole deterministisiä. Toistonappi tuottaa aina erilaisen vastauksen, eikä malli edes tiedä mitä se on tuottanut aikaisemmin.

Työssä Kielimallia ja RAG:ia käytettiin yhdessä tukemaan toistensa puutteita, mutta ne voivat toimia hyvin omillaan tarkoituksiin, missä ne voivat toimia parhaimmillaan.

RAG loisti yksinään erityisesti Unityn dokumentaation hakemisessa. Konteksteilla tai muilla kielisäännöillä ei ole niinkään väliä, jos haluat hakea tietyllä hakusanalla kaikki asiaan liittyvät dokumentaatiot. Ulkoasulla tai selvyydellä ei ole niinkään väliä, koska RAG osaa erotella koodit omiin upotteisiinsa ja eri dokumenttien sisällöt on eroteltu järkevästi.

Kielimallit ovat kieltämättä erinomaisia pelinkehittäjien apuna, mutta suurempi kysymys on paikallisten mallien arvo. Kaupallisten toimijoiden nauttima luottamus ei ole kovin suurta jatkuvien tietovuotojen tai tiedon väärinkäyttösyytösten takia. Suuret yritykset voivat saada huomattavaa arvoa, jos ne kehittävät oman mallinsa ja ajavat sitä paikallisesti. Turvaten näin yrityssalaisuutensa.

Suurimpia pienemmät yritykset voivat siltikin joutua turvautumaan kaupallisiin palveluihin, niiden kohtuuttomien kustannusten takia. Pienille tiimeille opinnäytetyön mukainen RAG + paikallinen AI yhdistelmä voi, jopa toimi parhaana vaihtoehtona, riippuen muista käytetyistä työkaluista.

Viimeisenä asiana on ehkä hieman eksistentiaalinen kriisi tekoälyn kehitysvauhdista. Opinnäytetyön tekemisen aikana (n, 4kk) joka ikinen käyttämäni tekniikka vanhentui ja uudemmat teknologiat korvasivat ne parempina tai kokonaan erilaisilla toimintatavoilla. RAG sai Knowledge Graph teknologian laajempaan käyttöön, LLama julkaisi kolmannen version tekoälymallistaan ja uusi tulokas Command R saa aina vaan parempia tuloksia koodin kanssa työskentelyssä.

LÄHTEET

Amaratunga, T. (2023) Understanding Large Language Models: Learning Their Underlying Concepts and Technologies. 1st edition. Berkeley, CA: Apress L. P.

Amit Y. (2023) Viitattu 25.4.2024 https://blog.fabrichq.ai/types-of-ai-agents-a-comprehensive-overview-with-examples-1e6e05d66e18

Cloudflare Vector database Viitattu 06.3.2024 https://www.cloudflare.com/en-gb/learning/ai/what-is-vector-database/

Facebook Research. FAISS Viitattu 30.4.2024 https://github.com/facebookresearch/faiss

Huggingface Blog. Mixtral moe architecture. Viitattu 22/1/24 https://huggingface.co/blog/moe

Meta AI blog. RAG. Viitattu 7.5.2024 https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/

Openai Enterprise. Viitattu 28.2.2024 https://openai.com/blog/introducing-chatgpt-enterprise

Politico. AI ruling. Viitattu 28.2.2024 https://www.politico.eu/article/ai-ruling-obstruct-british-efforts-protect-citizens-images-us-data-harvesting/

Reuters Legal. AI privacy paradox. Viitattu 15.3.2024 https://www.reuters.com/legal/legalindustry/privacy-paradox-with-ai-2023-10-31/

Sillytavern documentation. Vector context. Viitattu 8.3.2024 https://docs.sillytavern.app/extras/extensions/smart-context/

Techcrunch. Chatgpt privacy design. Viitattu 28.2.2024 https://techcrunch.com/2023/08/30/chatgpt-maker-openai-accused-of-string-of-data-protection-breaches-in-gdpr-complaint-filed-by-privacy-researcher/