Jari Antikainen

Paikallinen RAG-järjestelmä avoimilla kielimalleilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

9.4.2025

Tiivistelmä

Tekijä:	Jari Antikainen
Otsikko:	Paikallinen RAG-järjestelmä avoimilla kielimalleilla
Sivumäärä:	35 sivua
Aika:	9.4.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Tietotekniikka
Ohjaajat:	Osaamisaluejohtaja Janne Salonen

Insinöörityössä kehitettiin ja arvioitiin paikallisesti toimiva Retrieval-Augmented Generation (RAG) -järjestelmä. Tavoitteena oli rakentaa kysymys-vastaussovellus, joka hyödyntää avoimen lähdekoodin kielimalleja ja mahdollistaa organisaation omien dokumenttien käyttämisen tietolähteenä ilman riippuvuutta pilvipalveluista, parantaen näin tietosuojaa ja hallittavuutta.

Järjestelmän toteutus perustui LangChain-ohjelmistokehykseen. Paikallinen toiminta mahdollistettiin Ollama-sovelluksella käyttäen Llama 3.2 -kielimallia ja bge-m3-upotusmallia. Dokumenttien käsittelyssä ja indeksoinnissa hyödynnettiin Chroma-vektoritietokantaa. Tiedonhaussa sovellettiin kehittyneitä tekniikoita, kuten hybridihakua (BM25 ja semanttinen haku), monikyselyhakua sekä cross-encoder-pohjaista uudelleenjärjestämistä hakutulosten relevanssin parantamiseksi. Järjestelmälle toteutettiin myös interaktiivinen Streamlit-käyttöliittymä testausta varten.

Toteutettu RAG-järjestelmä osoittautui toimivaksi ja kykeni vastaamaan kysymyksiin paikallisesti tallennettujen dokumenttien perusteella. Kvalitatiivisessa testauksessa havaittiin, että kehittyneet hakutekniikat, erityisesti hybridihaku, parantavat vastausten tarkkuutta ja kattavuutta verrattuna perustason semanttiseen hakuun. Myös uudelleenjärjestämisen ja valinnaisen vastausten jatkojalostuksen vaikutuksia arvioitiin.

Työ osoitti, että paikallisten, avoimeen lähdekoodiin perustuvien RAG-järjestelmien rakentaminen on mahdollista ja ne tarjoavat käyttökelpoisen ratkaisun organisaatioille tietoturvalliseen tiedonhakuun omista aineistoista.

Avainsanat: tekoäly, suuret kielimallit, LLM, avoin lähdekoodi, Retrieval-Augmented Generation, RAG, LangChain, Ollama

---

Abstract

Author:	Jari Antikainen
Title:	Local RAG System with Open-Source Language Models
Number of Pages:	35 pages
Date:	9 April 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Information Technology
Supervisors:	Janne Salonen, Director of School

This Bachelor's thesis developed and evaluated a locally executable Retrieval-Augmented Generation (RAG) system. The objective was to build a question-answering application utilizing open-source language models, enabling the use of an organization's own documents as a knowledge source without reliance on cloud services, thus enhancing data privacy and control.

The system was implemented using the LangChain framework. Local execution was enabled by the Ollama server application, running the Llama 3.2 language model and the bge-m3 embedding model. A Chroma vector database was used for document processing and indexing. Advanced retrieval techniques were employed, including hybrid search (BM25 and semantic search), multi-query retrieval, and cross-encoder-based re-ranking to improve the relevance of search results. An interactive Streamlit-based user interface was also developed for testing purposes.

The implemented RAG system proved functional, capable of answering questions based on locally stored documents. Qualitative testing indicated that advanced retrieval techniques, particularly hybrid search, improved answer accuracy and coverage compared to basic semantic retrieval. The effects of re-ranking and optional answer refinement were also assessed.

The work demonstrated the feasibility of building local RAG systems based on open-source components, offering a viable solution for organizations needing secure information retrieval from their own data.

Keywords: AI, Large Language Models, LLM, Open Source, Retrieval-Augmented Generation, RAG, LangChain, Ollama

Sisällys

Lyhenteet

API: *Application Programming Interface*. Ohjelmointirajapinta. Määritelty joukko sääntöjä ja protokollia, joiden avulla eri ohjelmistokomponentit voivat kommunikoida keskenään ja hyödyntää toistensa toiminnallisuuksia.

GPT: *Generative Pre-trained Transformer*. Generatiivinen esikoulutettu Transformer-malli. Kielimallityyppi, joka perustuu Transformer-arkkitehtuuriin ja on esikoulutettu suurilla tekstiaineistoilla generoimaan ihmismäistä tekstiä.

LLM: *Large Language Model.* Suuri kielimalli. Tekoälymalli, joka on koulutettu tekstiaineistoilla ymmärtämään ja generoimaan luonnollista kieltä monimutkaisissa tehtävissä.

NLP: *Natural Language Processing*. Tietojenkäsittelytieteen ja tekoälyn ala, joka keskittyy tietokoneiden ja ihmiskielen väliseen vuorovaikutukseen, erityisesti kielen ymmärtämiseen, tulkintaan ja tuottamiseen.

OCR: *Optical Character Recognition*. Optinen tekstintunnistus. Tekstin muuntaminen kuvista tai asiakirjoista koneellisesti luettavaan muotoon.

RAG: *Retrieval-Augmented Generation*. Tekoälymenetelmä, jossa suuren kielimallin kykyä generoida vastauksia parannetaan hakemalla relevanttia tietoa ulkoisesta tietolähteestä.

1 Johdanto

Tekoälymallien, erityisesti suurten kielimallien (Large Language Models, LLM), kehitys on edennyt nopeasti viime vuosina luonnollisen kielen käsittelyssä. Näiden mallien kyky ymmärtää ja tuottaa ihmismäistä kieltä on avannut uusia mahdollisuuksia monilla sovellusalueilla, kuten asiakaspalvelussa, markkinoinnissa, tekstianalyysissä ja käännöksissä. Samalla yleisimmin käytetyt pilvipohjaiset ratkaisut ovat herättäneet kysymyksiä tietosuojasta, kustannuksista ja räätälöinnin rajoituksista.

Tämä insinöörityö sai alkunsa henkilökohtaisesta mielenkiinnosta ja tarpeesta tutkia vaihtoehtoisia lähestymistapoja. Paikallisesti asennettavat tekoälymallit ovat nousseet ajankohtaiseksi tutkimuskohteeksi, sillä ne tarjoavat potentiaalisia ratkaisuja edellä mainittuihin haasteisiin, erityisesti avoimen lähdekoodin mallien kehittymisen ja suosion kasvun myötä. Paikallinen toteutus mahdollistaa datan pysymisen organisaation hallinnassa ja tarjoaa suurempaa joustavuutta järjestelmien muokkaamiseen.

Insinöörityön tavoitteena oli selvittää, kuinka paikallisia, avoimen lähdekoodin kielimalleja voidaan soveltaa tiedon etsimiseen ja kysymyksiin vastaamiseen (Question-Answering, Q&A) organisaation omista dokumenttiaineistoista hyödyntäen Retrieval-Augmented Generation (RAG) -menetelmää. Lisäksi työssä pyrittiin arvioimaan paikallisten LLM-pohjaisten RAG-järjestelmien etuja ja rajoituksia verrattuna yleisempiin pilvipohjaisiin ratkaisuihin sekä kartoittamaan niiden käyttöönottoon liittyviä teknisiä ratkaisuja ja suorituskykyyn vaikuttavia tekijöitä.

Työn alussa käsitellään suurten kielimallien taustoja sekä suljettujen ja avoimen lähdekoodin mallien eroja (Luku 2). Tämän jälkeen perehdytään RAG-menetelmän periaatteisiin (Luku 3 ja 4). Työn käytännön osuudessa kuvataan kehitetyn paikallisen RAG-järjestelmän arkkitehtuuri ja toteutus (Luku 5). Lopuksi

esitetään päättelyosuutena yhteenveto saavutetuista tuloksista, tunnistetuista haasteista ja mahdollisista jatkokehityskohteista (Luku 6).

2 Suljettujen ja avoimen lähdekoodin kielimallien erot

Suurten kielimallien (LLM) kenttä on monimuotoinen, ja mallit sijoittuvat jatkumolle täysin suljetuista (proprietary) malleista avoimesti lisensoituihin ja jaettuihin malleihin (open-source). Useat suuret teknologiayritykset, kuten Meta, Google ja Mistral AI, kehittävät ja tarjoavat sekä kaupallisia suljettuja malleja, että avoimen lähdekoodin vaihtoehtoja. Tämä kaksoisstrategia vaikuttaa siihen, miten malleja voidaan käyttää, muokata ja ottaa käyttöön eri tarkoituksiin.

Tässä luvussa tarkastellaan näiden eri lähestymistapojen piirteitä. Aluksi käsitellään suljettujen mallien ominaisuuksia, jotka liittyvät usein keskitettyyn hallintaan ja kaupallisiin palveluihin (Luku 2.1). Tämän jälkeen perehdytään avoimen lähdekoodin malleihin, niiden etuihin ja siihen, miten myös suuret toimijat edistävät niiden kehitystä (Luku 2.2). Lisäksi tarkastellaan avoimien mallien saatavuuden kehitystä ja mahdollisuutta ajaa niitä paikallisesti (Luku 2.3) sekä kehitysyhteisöjen roolia innovaatioiden ja saavutettavuuden edistäjinä (Luku 2.4).

2.1 Suljetut kielimallit

Kehittyneimmät kielimallit, kuten OpenAI:n GPT-sarja, Anthropic Claude ja Google Gemini, ovat suljettuja järjestelmiä. Näitä käytetään tyypillisesti pilvipohjaisten selainkäyttöliittymien, sovellusten tai API-rajapintojen kautta, joita operoidaan massiivisissa datakeskuksissa. Näiden mallien vahvuutena pidetään usein niiden korkeaa ja tasaista suorituskykyä laajoissa tehtävissä, mikä perustuu patentoituun teknologiaan, valtaviin koulutusaineistoihin ja keskitettyyn hallintaan. Tämä voi vähentää virheiden riskiä ja varmistaa korkealaatuisen käyttökokemuksen monilla sovellusalueilla.

Vaikka näitä malleja tarjoavat yritykset saattavat julkaista myös avoimen lähdekoodin versioita, niiden kaupallinen kärki ja usein kaikkein suorituskykyisimmät

mallit pidetään suljettuina. Tämä keskittää vallan muutamien teknologiayritysten käsiin, jotka hallitsevat esimerkiksi:

- Toiminnallisuutta: Mitä tekoälymallit voivat tehdä ja mitä eivät (esim. sisällön sensuuri, erityisominaisuudet).
- Pääsyä: Kuka saa käyttää teknologiaa ja millä ehdoin (esim. API-avaimet, käyttöehdot).
- Kustannuksia: Hinnoittelumallia (usein token- tai käyttöpohjainen).
- Datan käyttöä: Miten käyttäjien syöttämää dataa käsitellään (yksityisyys ja tietoturva).

Tämä lähestymistapa herättää kysymyksiä liittyen tietoturvaan, yksityisyyteen ja eettisyyteen, kun dataa lähetetään ulkopuolisiin palveluihin. Lisäksi API-pohjainen hinnoittelu, joka perustuu usein käytettyjen tokeneiden, eli tekstin pienempien yksiköiden kuten sanojen ja merkkien määrään, voi muodostua kalliiksi suurilla datamäärillä tai intensiivisessä käytössä. Käyttäjien mahdollisuudet mukauttaa ja laajentaa näiden suljettujen mallien toimintaa ovat myös rajalliset verrattuna avoimiin vaihtoehtoihin.

2.2 Avoimen lähdekoodin kielimallit

Avoimen lähdekoodin kielimallit ovat nousseet keskeiseksi osaksi tekoälyn kehitystä. Merkittävä osa tästä kehityksestä tulee samoista suurista teknologiayrityksistä kuin suljetutkin mallit. Esimerkiksi Meta Llama-mallit (ja niiden päälle rakennetut versiot), Mistral AI -mallit, Google Gemma-mallit, sekä Alibaba Cloud Qwen-mallit ovat esimerkkejä laajasti käytetyistä ja suorituskykyisistä avoimen lähdekoodin malleista.

Tämän kehityksen merkittävyyttä korostaa myös pienempien toimijoiden, kuten kiinalaisen DeepSeekin, nousu. Vuoden 2025 alussa julkaistu DeepSeek R1 -kielimalli osoitti, että kilpailukykyistä suorituskykyä, joka vertautui johtaviin malleihin esimerkiksi matematiikassa ja koodauksessa, voidaan saavuttaa huomattavasti pienemmillä resursseilla. DeepSeekin mallin kehityskustannukset olivat vain murto-osa suurten toimijoiden investoinneista. DeepSeek ei ainoastaan julkaissut malliaan täysin avoimena, vaan myös kuvasi avoimesti sen

koulutusprosessin, mahdollistaen sen jatkokehittämisen eri käyttötarkoituksiin. [1] Tällaiset tehokkaat ja avoimet mallit voivat merkittävästi muuttaa markkinadynamiikkaa ja avata uusia mahdollisuuksia pienemmille toimijoille.

Avoimen lähdekoodin kielimallien strategista merkitystä Euroopassa korostaa OpenEuroLLM-hanke. [2] Tässä laajassa yhteistyöprojektissa joukko johtavia eurooppalaisia tekoäly-yrityksiä ja tutkimuslaitoksia, mukaan lukien suomalaiset AMD Silo AI, Turun yliopiston TurkuNLP-ryhmä, Helsingin yliopisto ja CSC, pyrkii yhdessä kehittämään seuraavan sukupolven avoimen lähdekoodin kielimalleja. [3] Hankkeen tavoitteena on vahvistaa Euroopan digitaalista itsemääräämisoikeutta ja parantaa sen kilpailukykyä globaaleilla tekoälymarkkinoilla, joita Yhdysvallat ja Kiina tällä hetkellä hallitsevat. [4] Keskeistä projektissa on sitoutuminen läpinäkyvyyteen ja sääntelyn noudattamiseen. Kehitettävät mallit noudattavat EU:n tekoälyasetuksen (AI Act) ja tietosuoja-asetuksen (GDPR) vaatimuksia, ja kaikki tuotettu koodi, data ja dokumentaatio julkaistaan avoimesti. [2; 5] Luomalla luotettavia, säädöstenmukaisia ja vapaasti saatavilla olevia korkealaatuisia tekoälyteknologioita OpenEuroLLM pyrkii madaltamaan kynnystä niiden käyttöönottoon erityisesti pk-yrityksissä ja julkisella sektorilla, tukien samalla monikielisyyttä ja kulttuurista monimuotoisuutta Euroopassa. [4; 5]

Avoimet mallit julkaistaan usein sallivilla lisensseillä, kuten MIT ja Apache 2.0, jotka mahdollistavat vapaan käytön, muokkauksen ja jopa kaupallisen hyödyntämisen tietyin ehdoin. Tämä avaa useita merkittäviä etuja:

1. Paikallinen käyttöönotto: Avoimia malleja voidaan ajaa omassa infrastruktuurissa (on-premise) tai paikallisilla työasemilla. Tämä on erityisen tärkeää organisaatioille, jotka käsittelevät arkaluonteista dataa (potilastiedot, liikesalaisuudet), sillä data ei poistu organisaation hallinnasta. Tämä tukee myös Edge Computing -ratkaisuja, joissa datan analysointi tapahtuu lähellä sen keräyspistettä, parantaen reagointikykyä ja vähentäen latenssia.

2. Muokattavuus ja hienosäätö: Kehittäjät voivat kouluttaa ja hienosäätää malleja omilla, tehtäväkohtaisilla aineistoillaan. Tämä tekee malleista

tehokkaampia kapeammissa erikoissovelluksissa verrattuna yleiskäyttöisiin, suuriin malleihin.

3. Kustannustehokkuus: Vaikka mallien ajaminen vaatii laskentatehoa, jatkuvien lisenssi- tai API-maksujen puuttuminen voi tuoda merkittäviä säästöjä verrattuna suljettuihin pilvipalveluihin. DeepSeekin osoittama kehitys- ja käyttökustannusten alhaisuus voi edelleen kiihdyttää tätä trendiä.

4. Läpinäkyvyys ja tutkimus: Avoimuus mahdollistaa mallien sisäisen toiminnan tutkimisen ja ymmärtämisen paremmin, mikä edistää alan tieteellistä tutkimusta ja kehitystä nopeammin.

5. Innovaatioiden demokratisointi: Avoimuus madaltaa kynnystä pienemmille yrityksille, tutkijoille ja yksittäisille kehittäjille osallistua tekoälyn kehittämiseen ja hyödyntämiseen, haastaen suurten toimijoiden keskittynyttä valtaa.

Avoimen lähdekoodin mallit vähentävät siis kustannuksia, parantavat tietoturvan hallintaa ja tarjoavat joustavuutta. Tämä on erityisen tärkeää organisaatioille, jotka käsittelevät sensitiivistä tai luottamuksellista tietoa, kuten potilastietoja, taloudellisia raportteja, liikesalaisuuksia tai yrityksen sisäistä viestintää. Vaikka avoimen lähdekoodin mallit voivat olla erittäin suorituskykyisiä, ovat ne usein kooltaan tai arkkitehtuuriltaan optimoituja siten, että niiden ajaminen paikallisesti on mahdollista kohtuullisella laitteistolla, toisin kuin kaikkein massiivisimmat suljetut mallit, jotka vaativat datakeskustason resursseja. Tämä tekee niistä erityisen houkuttelevia juuri paikallisiin RAG-järjestelmiin ja muihin sovelluksiin, joissa datan hallinta ja muokattavuus ovat tärkeitä.

Avoimen lähdekoodin yhteistyöluonne rohkaisee kollektiiviseen tekoälykehitykseen ja yhteistyöhön teknologiayhteisössä, mikä voi johtaa uusiin innovaatioihin nopeammin. Olemme todistamassa ratkaisevaa hetkeä tekoälyn kehityksessä: hallitseeko teknologiaa vain muutama vahva yritys, vai jakaantuuko kehitys ja

hyödyntäminen laajemmin. Avoimen lähdekoodin mallit ovat keskeinen tekijä tämän kehityksen suunnan kannalta.

2.3 Kehitys avoimien mallien saatavuudessa ja paikallisessa käytössä

Tekoälymallien teknologia on kehittynyt siihen pisteeseen, että yhä suorituskykyisempiä, avoimen lähdekoodin kielimalleja voidaan ajaa paikallisesti – kuluttajatason työasemilla, kannettavilla tietokoneilla ja joissain tapauksissa älypuhelimilla. Paikallisesti asennettavat tekoälymallit tarjoavat ratkaisun, joka ei ole riippuvainen internetissä toimivista palveluista tai ulkoisista API-rajapinnoista. Tämä on erityisen merkittävää tietosuojan ja luotettavuuden kannalta yksityisiä tai arkaluonteisia tietoja käsiteltäessä.

Parametrien määrä, joka kuvaa mallin oppimaa tietoa, ei ole ainoa suorituskyvyn mittari. Kapean toimialueen sovelluksissa ja esimerkiksi organisaation oman datan käsittelyssä mallin parametrisen muistin laajuudella ei ole yhtä suurta merkitystä kuin laajojen, yleisten kysymysten käsittelyssä.

Pienemmät mallit voivat olla erittäin tehokkaita monissa erityissovelluksissa, kuten pidempien tekstien yhteenvetojen tuottamisessa. Näiden mallien suorituskykyä voidaan lisäksi parantaa jatkokouluttamalla tai hienosäätämällä niitä tehtäväkohtaisesti. Hyvin hienosäädetty tai tehtävään sopiva malli voi olla tehokkaampi ja resurssiystävällisempi kuin valtava yleismalli.

On kuitenkin tilanteita, joissa jatkokouluttaminen ei ole tarkoituksenmukaisin ratkaisu. Esimerkiksi kysymys-vastausjärjestelmän rakentamisessa, jossa käyttäjien kysymyksiin vastataan organisaation omien dokumenttien pohjalta, kustannustehokkaampi tapa on hyödyntää lähestymistapaa, jossa tekoäly saa kerrallaan käsiteltäväkseen vain sen tiedon, joka on olennaista kysymyksiin vastaamiseksi (Retrieval-Augmented Generation, RAG). Tämä menetelmä paitsi parantaa vastausten tarkkuutta myös optimoi mallin resurssien käyttöä, sillä se käsittelee yksinomaan olennaista tietoa.

Lisäksi paikalliset mallit tarjoavat kustannusetuja, sillä niiden käyttö ei vaadi pilvipalveluiden jatkuvia lisenssimaksuja. Maksuton saatavuus tekee niistä houkuttelevan vaihtoehdon yksityishenkilöille, yrityksille ja akateemisille tutkijoille, jotka etsivät kustannustehokasta ja tietoturvallista tekoälyratkaisua.

OpenAI:n perustajajäsenen ja entisen pääsuunnittelija Ilya Sutskeverin mukaan nykyisen kaltaisen tekoälyn esikoulutuksen aikakausi tulee loppumaan. Aivan kuten öljy on rajallinen luonnonvara, internet sisältää rajallisen määrän ihmisen tuottamaa sisältöä. Olemme saavuttaneet huipputiedot, eikä niitä tule lisää. Meidän on käsiteltävä niitä tietoja, joita meillä on. Internet on vain yksi lähteistä. [6] AI-datan tulevaisuus tulee hajautetuilta ”reunoilta”. Edge Computing -tiedot ovat pohjimmiltaan erilaisia. Toisin kuin staattinen internet-data, Edge Computing -data on kontekstuaalista, reaaliaikaista tietoa ja ihmisten vuorovaikutusta laitteiden kanssa.

2.4 Kehitysyhteisöt: Yhteistyön ja innovaatioiden edistäjät

Avoimen lähdekoodin kehitysyhteisöt, erityisesti alustat kuten Hugging Face, ovat keskeisessä roolissa tekoälyteknologian kehityksessä ja levittämisessä. Ne tarjoavat paitsi pääsyn laajaan valikoimaan malleja (sekä yhteisön että suurten yritysten julkaisemia), myös työkaluja, kirjastoja ja dokumentaatiota niiden hyödyntämiseen. Hugging Facen Transformers-kirjasto on esimerkki laajalti käytetystä työkalusta, joka helpottaa vuorovaikutusta erilaisten kielimallien kanssa. [7]

Hugging Facen kaltaiset alustat mahdollistavat sen, että yhä useammat käyttäjät – opiskelijat, tutkijat, kehittäjät – voivat tutkia, kokeilla, luoda ja optimoida tekoälysovelluksia. Yhteisön jäsenet jakavat aktiivisesti omia toteutuksiaan, hienosäädettyjä malleja, tietojoukkoja ja parhaita käytäntöjä. Tämä yhteisöllinen toimintatapa nopeuttaa merkittävästi uusien ratkaisujen kehittämistä ja laajentaa tekoälyn saavutettavuutta, edistäen innovaatioita ja tiedon jakamista tekoälyn alalla.

3 Luonnollisen kielen käsittely ja suurten kielimallien kehitys

Luonnollisen kielen käsittely (Natural Language Processing, NLP) on tekoälyn osa-alue, joka keskittyy ihmiskielen koneelliseen käsittelyyn, analysointiin ja tuottamiseen. NLP:n avulla tietokoneet voivat ymmärtää, tulkita ja tuottaa ihmiskieltä tavalla, joka muistuttaa ihmisten välistä vuorovaikutusta. Sovelluskohteita löytyy laajasti, kuten puheentunnistuksesta, tekstianalyysistä, konekääntämisestä ja virtuaaliavustajien toiminnasta. Viime vuosina erityisesti suurten kielimallien (LLM) kehitys on mullistanut NLP-alaa.

Suuret kielimallit ovat kehittyneitä neuroverkkoihin perustuvia järjestelmiä, jotka hyödyntävät syväoppimista laajojen tekstiaineistojen käsittelyyn. Mallien koulutusprosessissa käytetään valtavia tekstimääriä, joita kutsutaan koulutusdataksi. Aineistot koostuvat kirjoista, artikkeleista ja verkkosivustoilta kerätyistä digitaalisessa muodossa olevista tekstilähteistä, ja niiden perusteella malli oppii sanojen, lauseiden ja niiden välisten suhteiden merkitykset. Mitä suuremman ja tarkemman tietokannan kielimalli saa käyttöönsä ja mitä kauemmin sen annetaan oppia, sitä paremmin se kykenee ennustamaan ja tuottamaan tekstiä. Suurten kielimallien kehitys on muuttanut radikaalisti tapaa, jolla NLP-sovellukset toimivat, mahdollistaen entistä luonnollisemman ja tehokkaamman vuorovaikutuksen tietokoneiden ja ihmisten välillä.

3.1 Transformer-arkkitehtuuri

Merkittävä edistysaskel luonnollisen kielen käsittelyn (NLP) kehityksessä saavutettiin vuonna 2017, kun Googlen tutkijat julkaisivat Transformer-arkkitehtuurin tieteellisessä artikkelissa "Attention Is All You Need". [8; 9] Tämä arkkitehtuuri mullisti alan, sillä se mahdollisti tehokkaan tiedon rinnakkaiskäsittelyn, pystyi mallintamaan sanojen välisiä pitkän matkan riippuvuuksia tehokkaasti ja ratkaisi aiempien mallien, kuten RNN:n ja LSTM:n, skaalautuvuuteen ja suorituskykyyn liittyviä haasteita. Transformer-arkkitehtuurista tuli nopeasti perusta monille suurille kielimalleille (LLM), mahdollistaen yhä suurempien ja kyvykkäämpien mallien kehittämisen.

Yksi näkyvimmistä toimijoista tämän teknologian hyödyntämisessä on ollut OpenAI. Yrityksen Generative Pre-trained Transformer (GPT) -mallisarja alkoi alkuperäisellä GPT-mallilla vuonna 2018 (n. 117 miljoonaa parametria) ja jatkui GPT-2:lla vuonna 2019 (n. 1,5 miljardia parametria), joka oli viimeinen laajasti avoimena MIT-lisenssillä julkaistu versio. [10; 11] Merkittävä harppaus tapahtui vuonna 2020 julkaistun GPT-3:n myötä (175 miljardia parametria). Se ei ollut ainoastaan huomattavasti edeltäjiään suurempi, vaan myös merkittävästi monipuolisempi malli. GPT-3 osoitti kykyä suoriutua laajasta kirjosta erilaisia kielitehtäviä, kuten tekstin generointia, kääntämistä, yhteenvetojen tekemistä ja jopa yksinkertaista koodin kirjoittamista, mikä oli selvä parannus aiempiin malleihin. [10; 11]

Vaikka suuret kielimallit olivat jo olemassa pitkään tutkimus- ja kehityskäytössä, niin suuren yleisön tietoisuuteen ja laajaan käyttöön ne nousivat räjähdysmäisesti OpenAI:n julkaistua käyttäjäystävällisen ChatGPT-chatbotin marraskuussa 2022. Tämä palvelu, joka perustui edelleen kehitettyyn GPT-3.5-malliin ja oli hienosäädetty keskustelemaan ihmisten kanssa, saavutti ennennäkemättömän suosion. Se keräsi sata miljoonaa käyttäjää alle kahdessa kuukaudessa. Ironista kyllä, OpenAI:n sisällä odotukset eivät olleet korkealla ennen julkaisua. Yrityksen entinen päätutkija Ilya Sutskever on myöhemmin myöntänyt, että hän ei pitänyt ChatGPT:tä erityisen hyvänä sen taipumuksen vuoksi antaa vääriä vastauksia faktakysymyksiin ja pelkäsi sen olevan jopa "tylsä". [12; 13] Suosion avaimeksi osoittautui kuitenkin helppokäyttöinen käyttöliittymä ja ilmaisuus. ChatGPT:n alkuperäinen tarkoitus oli kerätä käyttäjäpalautetta mallin kehittämiseksi. Tässä hyödynnettiin RLHF-tekniikkaa (Reinforcement Learning from Human Feedback), jolla mallille opetettiin ihmiskäyttäjien suosimia vastauksia. [12] Palvelu teki generatiivisesta tekoälystä maailmanlaajuisen ilmiön ja muutti pysyvästi käsitystä siitä, mihin generatiivinen tekoäly voi pystyä.

3.2 Suurten kielimallien haasteet

Vaikka suuret kielimallit ovat saavuttaneet merkittäviä edistysaskelia luonnollisen kielen käsittelyssä, niiden käyttöön liittyy myös useita haasteita ja

rajoituksia. Yksi merkittävimmistä haasteista on tuotetun sisällön tarkkuus ja luotettavuus. Koska kielimallien tuottama tieto voi olla vain yhtä luotettavaa kuin niiden koulutusdata, puutteellinen, virheellinen tai vanhentunut koulutusaineisto voi johtaa epätarkan tai jopa harhaanjohtavan sisällön tuottamiseen.

Toinen keskeinen haaste liittyykin koulutusdatan ajantasaisuuteen. Suurten kielimallien koulutusdata on väistämättä aina jossain määrin vanhentunutta. Mallit koulutetaan tietyllä hetkellä kerätyllä data-aineistolla, minkä vuoksi niiden "tietämyksellä" on rajapäivämäärä. Jos malli on koulutettu tietoaineistoilla syyskuuhun 2024 asti, se ei tiedä mitään sen jälkeisistä tapahtumista. Tämä tarkoittaa, että käyttäjien kyselyt voivat koskea tapahtumia tai tietoja, joita mallilla ei ole saatavilla. Tämä voi olla ongelma sovelluksissa, jotka vaativat pääsyä tuoreimpaan tietoon, kuten uutisointi, reaaliaikainen data-analyysi tai ajankohtaiset tapahtumat. Vaikka malli osaisi vastata kysymykseen ”Kuka on Suomen pääministeri?”, sen tieto voi olla vanhentunut, jos pääministeri on vaihtunut koulutusdatan jälkeen.

Lisäksi suurten kielimallien toimintaa rajoittaa niin sanottu konteksti-ikkuna (context window). Malli voi käsitellä kerrallaan vain rajallisen määrän tietoa. Jos relevantti tieto ylittää tämän rajan tai jos kysymys on monimutkainen ja vaatii laajempaa kontekstia kuin mallin rajalliseen "työmuistiin" mahtuu, malli voi tuottaa virheellistä, epärelevanttia tai jopa täysin keksittyä tietoa. Näissä olosuhteissa mallit ovat taipuvaisia hallusinoimaan. Ne tuottavat vastauksia, jotka vaikuttavat uskottavilta mutta eivät perustu todellisuuteen tai annettuun rajalliseen kontekstiin.

Näiden haasteiden vuoksi on kehitetty menetelmiä, joilla pyritään parantamaan suurten kielimallien tarkkuutta, luotettavuutta ja ajankohtaisuutta. Yksi merkittävä lähestymistapa on Retrieval-Augmented Generation (RAG). RAG pyrkii parantamaan kielimallien luotettavuutta, tarkkuutta ja ajantasaisuutta yhdistämällä kielimallin generatiiviset kyvyt ulkoisiin, ajantasaisiin tietolähteisiin perustuvaan tiedonhakuun. Tämän avulla mallin vastauksia voidaan parantaa ja päivittää

ilman, että itse kielimallia tarvitsee jatkuvasti uudelleenkouluttaa. RAG:n toimintaperiaatetta käsitellään tarkemmin seuraavassa luvussa.

4 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) on menetelmä, joka tehostaa suurten kielimallien (LLM) suorituskykyä yhdistämällä ne ulkoisiin tietolähteisiin. Se esiteltiin ensimmäisen kerran vuonna 2020 Facebook AI Researchin (nyk. Meta AI) tutkijoiden toimesta julkaisussa ”Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. [14; 15] Sen sijaan, että LLM tukeutuisi ainoastaan sisäiseen, staattiseen koulutusdataansa, RAG mahdollistaa ajantasaisen ja kontekstisidonnaisen tiedon hyödyntämisen vastausten generoinnissa. Tämä parantaa vastausten tarkkuutta ja relevanssia, sekä vähentää hallusinointia eli virheellisen tiedon tuottamista.

4.1 RAG:n yleiskuvaus

Yksinkertaistettuna RAG-menetelmän toimintalogiikka voidaan jakaa kolmeen keskeiseen vaiheeseen, jotka heijastuvat myös menetelmän nimessä: [16]

1. Haku (Retrieval): Kun järjestelmä saa käyttäjältä kyselyn, ensimmäinen vaihe on etsiä siihen liittyvää relevanttia tietoa määritellyistä ulkoisista tietolähteistä (esim. dokumentit, tietokannat, API-rajapinnat). Tavoitteena on löytää ne tiedon osat, jotka todennäköisimmin vastaavat kyselyyn tarkasti.

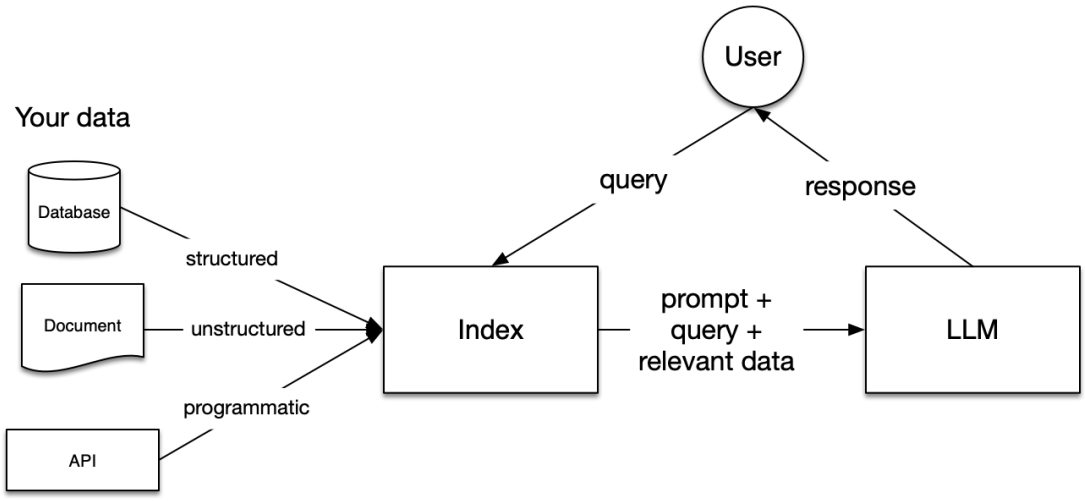
2. Rikastaminen (Augmentation): Tässä vaiheessa haettu tieto yhdistetään alkuperäiseen kyselyyn, luoden kielimallille (LLM) rikastettu kehote (enriched prompt). Tämä toimii LLM:n kontekstina ja ohjaa sitä tuottamaan vastauksen.

3. Generointi (Generation): Lopuksi LLM prosessoi rikastetun kehotteen ja generoi lopullisen vastauksen käyttäjälle. Koska LLM:llä on nyt

käytössään relevanttia ulkoista tietoa, sen tuottama vastaus perustuu sille annettuun, ajantasaiseen tai spesifiseen tietoon mallin koulutusdatan sijaan.

Vaikka haku, rikastaminen ja generointi kuvaa RAG:n ydinprosessin ja termit, käytännön RAG-järjestelmien toteutus vaatii myös muita, valmistelevia ja tukevia toimintoja.

Keskeisin näistä toimenpiteitä on Indeksointi (Indexing), jossa ulkoiset tietolähteet käsitellään (esim. pilkotaan osiin, luodaan upotukset) ja tallennetaan hakukelpoiseen muotoon. Kuva 1 havainnollistaa RAG-järjestelmän kyselyvaiheen toimintamallia, joka hyödyntää tätä ennalta tehtyä indeksointia. Laajemmin tarkasteltuna RAG-sovelluksen toiminta sisältää vaiheita datan lataamisesta ja indeksoinnista aina kyselyyn ja järjestelmän arviointiin, kuten esimerkiksi LlamaIndex-ohjelmistokehyksen viisivaiheisessa mallissa: datan lataus, indeksointi, tallennus, kysely ja arviointi. [17]



Kuva 1. Tyypillinen RAG-järjestelmän kyselyvaiheen toimintamalli, joka hyödyntää ennalta ladattua, indeksoitua ja tallennettua dataa. Prosessi sisältää käyttäjän kyselyn (query), relevantin tiedon haun (Retrieval) indeksistä (Index), haetun tiedon yhdistämisen kehotteeseen (Augmentation, prompt + query + relevant data) ja vastauksen generoinnin (Generation) kielimallilla (LLM). [17]

Tämä prosessi mahdollistaa LLM:n tuottamaan tarkempia ja ajantasaisempia vastauksia erityisesti silloin, kun tarvitaan yksityiskohtaista tai erikoistunutta tietoa, jota ei löydy kielimallin esikoulutuksesta. RAG-järjestelmistä on olemassa sekä yksinkertaisia perusratkaisuja että kehittyneempiä versioita.

4.2 Naiivi RAG

Naiivi RAG (Naive RAG) edustaa RAG-arkkitehtuurin perusmuotoa. Se toteuttaa ydinkomponentit suoraviivaisesti ilman monimutkaisempia optimointeja. Sen keskeiset piirteet ovat:

1. Indeksointi: Dokumentit pilkotaan tyypillisesti kiinteän kokoisiin osiin, ja niille luodaan upotukset ilman syvällisempää metadatan käsittelyä tai hierarkkista jäsentelyä.

2. Hakeminen: Käytetään yleensä perustason vektoripohjaista samankaltaisuushakua (esim. kosinisamankaltaisuus) relevanttien tieto-osioiden löytämiseksi. Hakua ei yleensä suodateta tai optimoida enempää.

3. Kyselyn käsittely: Käyttäjän alkuperäinen kysely välitetään sellaisenaan haku- ja generointivaiheisiin ilman muokkauksia tai laajennuksia.

4. Kontekstin käsittely: Haetut tieto-osiot yhdistetään suoraan kyselyyn ja syötetään LLM:lle ilman uudelleenjärjestelyä, tiivistämistä tai tarkempaa analyysiä niiden relevanssista.

5. Kehotepohjan muotoilu (Prompt Engineering): Käytössä on usein kiinteä, ennalta määritelty kehotepohja (prompt template), johon haettu konteksti ja kysely sijoitetaan. Pohja ei muutu dynaamisesti.

6. Generointi: LLM tuottaa vastauksen yhdellä kertaa annetun kontekstin ja kyselyn perusteella ilman monivaiheista prosessointia tai iteratiivista parantelua.

Vaikka naiivi RAG on konseptina yksinkertainen ja helppo toteuttaa, sen suorituskyky voi vaihdella merkittävästi ja kohdata useita haasteita riippuen datan laadusta ja kyselyiden monimutkaisuudesta. Naiivin mallin hakuvaiheen rajoitukset, kuten perustason samankaltaisuushaku ilman syvällisempää kontekstin ymmärrystä, voivat johtaa epäolennaisten tai laadultaan vaihtelevien tieto-osioiden noutamiseen, jolloin kriittistä informaatiota voi jäädä puuttumaan. Lisäksi haun ja generoinnin välinen integraatio on naiivissa mallissa huonompi, jos haettu konteksti syötetään LLM:lle sellaisenaan ilman tarkempaa relevanssin arviointia tai suodatusta. Tämä voi johtaa generointivaiheen ongelmiin, kuten hallusinointiin tai epätarkkoihin vastauksiin, vaikka ulkoista tietoa onkin tarjolla.

4.3 Edistyneet RAG-ratkaisut

Tarpeen mukaan RAG-järjestelmiä voidaan kehittää huomattavasti monipuolisemmiksi ja tehokkaammiksi. Edistyneet RAG-ratkaisut sisältävät usein seuraavia parannuksia naiiviin lähestymistapaan verrattuna:

1. Monipuolisempi indeksointi: Pelkän kiinteäkokoisen pilkkomisen sijaan dokumenttien käsittely voi olla dynaamisempaa käsiteltävän datan perusteella. Indeksoinnissa voidaan hyödyntää rikasta metatietoa (esim. päiväys, lähde, avainsanat) ja luoda hierarkkisia rakenteita, jotka mahdollistavat tarkemman ja kohdennetumman haun kuin naiivi RAG.

2. Monipuolisempi haku ja jälkikäsittely: Yksinkertaisen vektorihakua voidaan täydentää monivaiheisilla strategioilla. Esimerkiksi hybridihaku yhdistää semanttisen ja avainsanapohjaisen haun vahvuudet. Hakutuloksia voidaan myös aktiivisesti suodattaa, järjestellä uudelleen relevanssin mukaan (re-ranking) tai kyselyä automaattisesti muokata parempien tulosten saamiseksi.

3. Älykkäämpi kyselyn käsittely: Alkuperäistä kyselyä (query) voidaan analysoida ja laajentaa LLM:n avulla (query expansion) tai tarkentaa

metadatan perusteella, jotta se vastaa paremmin käyttäjän todellista tiedontarvetta, vähentäen monitulkintaisuutta.

4. Aktiivinen kontekstin käsittely: Sen sijaan, että kaikki haetut tieto-osiot syötetään suoraan LLM:lle, edistynyt RAG voi käsitellä kontekstia aktiivisesti: poistaa epärelevantteja osioita, tiivistää tietoa tai järjestää palaset loogisemmin, parantaen LLM:lle syötettävän tiedon laatua.

5. Kehotepohjan muotoilu: Kehotepohjat (prompt template) voivat olla dynaamisia ja mukautua haetun tiedon ja alkuperäisen kyselyn perusteella, mikä parantaa generoinnin tarkkuutta ja ohjaa LLM:ää paremmin.

6. Iteratiivinen generointi: Vastausprosessi voi olla monivaiheinen. Järjestelmä voi käyttää useita haku- ja generointikierroksia, joissa tuloksia tarkennetaan asteittain, tuottaen lopulta laadukkaamman vastauksen kuin yhdellä kertaa generoitu.

7. Mallien käyttö: Edistynyt RAG voi hyödyntää eri kielimalleja tai mallikonfiguraatioita eri prosessointivaiheissa (esim. yksi uudelleenjärjestelyyn (re-ranking) kehitetty malli, toinen lopullisen vastauksen generointiin), mikä mahdollistaa resurssien optimoinnin ja joustavamman järjestelmän.

Edistyneet RAG-ratkaisut tarjoavat parempaa tarkkuutta, relevanssia ja joustavuutta verrattuna naiiviin malliin. Niiden toteutus vaatii kuitenkin enemmän asiantuntemusta järjestelmän eri osien kehittämiseen ja integrointiin, sekä enemmän kehitystyöhön käytettävää aikaa ja resursseja. Edistyneet menetelmät ovat perusteltuja erityisesti silloin, kun tarvitaan suurta tarkkuutta, käsitellään monimutkaista tai laajaa tietoaineistoa tai kun käyttäjien kyselyt ovat monitulkintaisia – esimerkiksi vaativissa asiantuntijatehtävissä, kuten lääketieteellisen tiedon tai juridisten dokumenttien analysoinnissa.

5 Toteutus: Paikallisen RAG-järjestelmän kehittäminen

Tämän insinöörityön käytännön osuudessa kehitettiin ja toteutettiin paikallisesti toimiva Retrieval-Augmented Generation (RAG) -järjestelmä. Tavoitteena oli rakentaa kysymys-vastaussovellus (Q&A), joka hyödyntää avoimen lähdekoodin suuria kielimalleja (LLM) ja kehittyneitä tiedonhakutekniikoita ilman riippuvuutta pilvipalveluista. Tämä mahdollistaa järjestelmän käytön paikallisissa ympäristöissä, joissa datan yksityisyys ja tietoturva ovat ensisijaisia.

5.1 Teknologiavalinnat ja arkkitehtuuri

Paikallisen RAG-järjestelmän toteutuksen perustana oli mahdollistaa suurten kielimallien (LLM) ja upotusmallien suorittaminen paikallisesti ilman riippuvuutta ulkoisista pilvipalveluista. Tämä edellytti teknologiavalintoja, jotka tukevat paikallista toimintaa ja helppoa integroitavuutta. Järjestelmän arkkitehtuuri rakentui seuraavien pääkomponenttien varaan:

1. Ollama – paikallinen mallipalvelin: Koko järjestelmän ytimessä ja ensimmäisenä vaatimuksena oli kyky ajaa kielimalleja paikallisesti. Tähän tehtävään valittiin Ollama-sovellus. [18] Se on komentorivipohjainen sovellus, joka yksinkertaistaa kielimallien ja upotusmallien lataamista, hallintaa ja ajamista omalla tietokoneella.

   Kun Ollama-palvelu on käynnissä, se tarjoaa paikallisen HTTP-pohjaisen API-rajapinnan, tyypillisesti osoitteessa ”http://localhost:11434”. [18] Kaikki RAG-järjestelmän muut komponentit, jotka tarvitsevat kielimallin toiminnallisuuksia (kuten tekstin generointia tai upotusten luontia), kommunikoivat mallien kanssa lähettämällä API-kutsuja tähän paikalliseen Ollama-päätepisteeseen. Esimerkiksi vastausten generointi tapahtuu POST-pyynnöllä osoitteeseen ”/api/chat” ja upotusten luonti POST-pyynnöllä osoitteeseen ”/api/embed”. Tässä työssä Ollaman kautta ajettiin Meta Llama 3.2 -kielimalleja, kokeillen 1 ja 3 miljardin parametrin versioita (1B ja 3B), ja BAAI bge-m3-upotusmallia. [18; 19; 20; 21]

2. LangChain-ohjelmistokehys: RAG-sovelluksen rakentamiseen ja eri komponenttien yhteenliittämiseen valittiin Python-pohjainen LangChain-ohjelmistokehys. [22] Sen modulaarinen rakenne mahdollisti RAG-putken eri vaiheiden (dokumenttien lataus, pilkkominen, upotusten luonti, haku, vastausten generointi) joustavan toteuttamisen. Lisäksi LangChain mahdollisti erilaisten tekniikoiden, kuten hybridihakujen ja uudelleenjärjestäjien (re-rankers), kokeilemisen ja integroimisen tehokkaasti. Vaikka tarjolla on muitakin vastaavia ohjelmistokehyksiä (framework), kuten LlamaIndex, LangChainin laaja ekosysteemi ja joustavuus eri komponenttien yhdistelyssä koettiin tämän työn luonteeseen sopivammaksi. LangChain tarjoaa valmiita integraatioita lukuisiin työkaluihin, mukaan lukien Ollama. LangChainin komponentit, kuten ChatOllama (kielimallille) ja OllamaEmbeddings (upotusmallille), kommunikoivat paikallisen Ollama API-rajapinnan kanssa. [22]

3. Chroma-vektoritietokanta: Tekstidokumenttien pilkottujen osien (chunks) ja niitä vastaavien numeeristen upotusten tallentamiseen valittiin Chroma-vektoritietokanta. [23] Se on avoimen lähdekoodin vektoritietokanta, joka on suunniteltu erityisesti tekoälysovellusten tarpeisiin. Chroma DB:n etuina tässä työssä olivat sen helppo käyttöönotto paikallisesti, kyky tallentaa data pysyvästi levylle ja integraatio LangChainin kanssa. [23]

4. Streamlit-käyttöliittymä: Järjestelmälle toteutettiin interaktiivinen selainpohjainen käyttöliittymä käyttäen Streamlit-kirjastoa. [24] Streamlit mahdollistaa nopean ja helpon tavan rakentaa datavetoisia selainsovelluksia Pythonilla. Tässä työssä sen avulla luotiin chat-tyylinen käyttöliittymä, jossa käyttäjä voi esittää kysymyksiä, nähdä vastaukset ja säätää RAG-järjestelmän parametreja. [24]

Tämä modulaarinen arkkitehtuuri, jonka keskiössä on paikallinen Ollama-sovellus, mahdollisti joustavan paikallisen RAG-järjestelmän toteuttamisen.

5.2 Datan esikäsittely ja indeksointi

RAG-järjestelmän kyky tuottaa tarkkoja ja relevantteja vastauksia riippuu olennaisesti sille syötetyn datan laadusta ja soveltuvuudesta. Ennen teknistä toteutusta on kriittistä tunnistaa ja valita ne datalähteet, jotka sisältävät tarvittavan tiedon käyttäjien todennäköisiin kysymyksiin vastaamiseksi. Vaikka tässä insinöörityössä käytettiin vain muutamaa valittua PDF-dokumenttia testidatana, todellisissa organisaatioympäristöissä datan laatuun liittyvät haasteet korostuvat. Data voi olla vanhentunutta, sisältää päällekkäisyyksiä, olla epäjohdonmukaisesti muotoiltua tai sisältää irrelevanttia tietoa. Datan huolellinen kuratointi ja valmistelu ovatkin ensiarvoisen tärkeitä RAG-järjestelmän tehokkuuden varmistamiseksi.

Kun datalähteet oli valittu, itse esikäsittely- ja indeksointiprosessi (ingestion, indexing) tässä työssä toteutettiin seuraavien vaiheiden kautta, jotta paikallinen LLM pystyi hyödyntämään dokumenttien sisältöä tehokkaasti:

1. Dokumenttien lataus: Lähdedokumentit (PDF) ladattiin koodissa määritellystä hakemistosta. Työssä hyödynnettiin LangChainin PDF Directory Loader -komponenttia (PyPDFDirectoryLoader).

2. Tekstin pilkkominen (Chunking): Ladatut dokumentit pilkottiin pienempiin, hallittavampiin osiin (chunks). Tässä käytettiin LangChainin RecursiveCharacterTextSplitteriä, joka pyrkii säilyttämään tekstin semanttisen yhtenäisyyden jakamalla sen ensisijaisesti kappaleiden ja lauseiden väleistä. Optimaalisen palakoon (chunk_size) ja päällekkäisyyden (chunk_overlap) löytämiseksi tehtiin kokeiluja. Tavoitteena oli säilyttää riittävä konteksti kussakin palassa, mutta pitää palat samalla riittävän pieninä upotusmallin ja kielimallin konteksti-ikkunan rajoitusten vuoksi. Itse kehitetty kokeellinen EnhancedDocumentLoader-moduuli pyrki myös tunnistamaan taulukoita ja muuntamaan ne Markdown-muotoon, jotta LLM voisi käsitellä niitä paremmin, pitäen taulukot mahdollisuuksien mukaan yhtenäisinä paloina.

3. Metadatan luonti: Jokaiselle tekstipalalle (chunk) generoitiin jäsenneltyä metatietoa käyttäen paikallista Llama 3.2 -kielimallia Ollaman kautta (ollama_metadata_tagger-funktio). Tämä rikastettu metadata sisälsi esimerkiksi dokumentin alkuperäisen otsikon, automaattisesti tunnistettuja avainsanoja, numeerisen datan esiintymisen, mainitut nimet (henkilöt, organisaatiot) ja dokumenttityypin. Tämän metadatan tarkoitus oli mahdollistaa myöhemmässä hakuvaiheessa tarkempi ja kohdennetumpi tiedonhaku. Lisäksi jokaiseen palaan liitettiin perusmetatiedot, kuten lähdetiedoston nimi ja sivunumero.

4. Upotusten luonti (Embedding): Tekstipalat ja niiden metadata muunnettiin numeerisiksi vektoreiksi eli upotuksiksi. Tässä käytettiin Ollaman kautta ajettavaa bge-m3-upotusmallia (OllamaEmbeddings). Nämä upotukset kuvaavat palojen semanttista merkitystä, mahdollistaen samankaltaisen sisällön löytämisen vektorihakujen avulla.

5. Vektoritietokantaan tallennus: Luodut upotukset ja niihin liittyvä tekstisisältö sekä metadata tallennettiin paikalliseen Chroma-vektoritietokantaan. Järjestelmään toteutettiin myös logiikka, joka tarkistaa datakansion muutokset. Se tunnistaa uudet lisättävät tiedostot ja poistaa hakemistosta poistetut dokumentit myös vektoritietokannasta. Lisäykset ja poistot tehdään eräajona (batch processing) tehokkuuden parantamiseksi. Jokaiselle tekstipalalle generoidaan yksilöllinen tunniste, joka perustuu mm. lähdetiedostoon, sivunumeroon, palanumeroon ja sisällön tiivisteeseen, jotta päivitysten hallinta olisi luotettavampaa.

Tämä vaiheittainen prosessi loi perustan tiedonhaulle ja sitä kautta koko RAG-järjestelmän toimivuudelle.

5.3 Tiedonhaku (Retrieval)

Tiedonhaun tehokkuus ja tarkkuus ovat keskeisiä RAG-järjestelmän toiminnan kannalta. Kun käyttäjä esittää kyselyn, järjestelmän tulee löytää indeksoidusta

datasta ne olennaisimmat tekstipalat (chunks), jotka sisältävät vastauksen kannalta relevanttia tietoa. Tässä työssä keskityttiin kehittyneisiin hakutekniikoihin perinteisen semanttisen haun täydentämiseksi ja parantamiseksi.

5.3.1 Hybridihaku (Hybrid Search)

Pelkkä semanttinen haku, joka perustuu upotusten samankaltaisuuteen, ei aina tunnista täydellisesti kyselyitä, jotka sisältävät tarkkoja avainsanoja, nimiä tai termejä. Tämän vuoksi toteutettiin hybridihaku, joka yhdistää semanttisen haun vahvuudet perinteisempään avainsanapohjaiseen hakuun. [25] Tässä työssä käytettiin BM25-algoritmia avainsanapohjaisena hakuna. Nämä kaksi hakutapaa – semanttinen vektorihaku ja BM25 – yhdistettiin LangChainin EnsembleRetriever-komponentilla. [26] Tämä komponentti mahdollistaa eri hakijoiden (retrievers) tulosten yhdistämisen ja painottamisen. Painoarvoja (weights) säätämällä voidaan hakea tasapainoa semanttisen relevanssin ja tarkan termivastaavuuden välillä (Esimerkkikoodi 1).

```
def create_hybrid_retriever(vector_store, documents, k, vector_w):
     # Create vector store retriever
     vectorstore_retriever = vector_store.as_retriever(
          search_type="similarity",
          search_kwargs={"k": k}
     )

     # Create BM25 retriever
     bm25_retriever = BM25Retriever.from_documents(documents)
     bm25_retriever.k = k

     # Create ensemble retriever
     ensemble_retriever = EnsembleRetriever(
          retrievers=[vectorstore_retriever, bm25_retriever],
          weights=[vector_w, 1.0 - vector_w] # sum should be one
     )

     return ensemble_retriever
```

Esimerkkikoodi 1. Hybridihaun toteutus Python-funktiolla. Funktio luo EnsembleRetriever-olion, joka yhdistää semanttisen vektorihakijan (vectorstore_retriever) ja avainsanapohjaisen BM25-hakijan (bm25_retriever). Hakijoiden painoarvoja (weights) voidaan säätää (tässä vector_w vektorihakijalle ja 1.0 - vector_w BM25-hakijalle), ja niiden summan tulee olla 1.0.

5.3.2 Monikyselyhaku (Multi-Query Retrieval)

Joskus käyttäjän alkuperäinen kysely voi olla monitulkintainen tai liian yleinen, jolloin pelkkä hybridihakukaan ei välttämättä löydä kaikkea relevanttia tietoa. Monikyselyhaku pyrkii ratkaisemaan tämän ongelman. [27] Siinä kielimallia (LLM) käytetään generoimaan alkuperäisen kyselyn pohjalta useita rinnakkaisia, hieman eri näkökulmista muotoiltuja kyselyitä. Tämän jälkeen haku suoritetaan kaikilla näillä generoiduilla kyselyillä, mukaan lukien alkuperäinen kysely, käyttäen pohjana edellä luotua hybridihakijaa. Tulokset yhdistetään, mikä laajentaa haun kattavuutta ja parantaa mahdollisuuksia löytää relevanttia tietoa, vaikka alkuperäinen kysely olisikin ollut epätarkka. Toteutuksessa käytettiin LangChainin MultiQueryRetriever-komponenttia, joka hyödynsi pohjana edellä kuvattua hybridihakijaa (Esimerkkikoodi 2). [28]

```
# Create base_retriever using create_hybrid_retriever
base_retriever = create_hybrid_retriever(
     vector_store,
     documents,
     k=st.session_state.retriever_k,
     vector_w=st.session_state.vector_weight
)

# Define the multi-query template
multi_query_template = """
Based on the following question, generate three specific, concise var-
iations of the question that aim to retrieve the most relevant infor-
mation from the provided context only.

Original Question: {question}
"""

# Create multi-query retriever
retriever = MultiQueryRetriever.from_llm(
     retriever=base_retriever,
     llm=llm,
     include_original=True,
     prompt=query_generation_prompt
)

query_generation_prompt = PromptTemplate.from_teplate(multi_query_tem-
plate)

# [...] Rest of the function [...]
```

Esimerkkikoodi 2. Monikyselyhakijan (MultiQueryRetriever) alustus query_data -funktiossa. Se käyttää pohjana olevaa hybridihakijaa (base_retriever), kielimallia (llm) ja query_generation_prompt-kehotepohjaa luodakseen ja suorittaakseen useita hakukyselyitä alkuperäisen kyselyn (question) laajentamiseksi.

5.3.3 Uudelleenjärjestäminen (Re-ranking)

Sekä hybridi- että monikyselyhaku voivat palauttaa suuren määrän dokumenttipätkiä, joista kaikki eivät välttämättä ole yhtä relevantteja lopullisen vastauksen kannalta. Uudelleenjärjestäminen on vaihe, jossa alkuperäisen haun palauttamat tulokset arvioidaan tarkemmin ja järjestetään uudelleen relevanssin perusteella. [25; 29]

Tässä työssä käytettiin erillistä, tähän tehtävään optimoitua cross-encoder-mallia (BAAI/bge-reranker-v2-m3). [30] Cross-encoder-malli arvioi kyselyn ja jokaisen palautetun dokumentin välistä relevanssia tarkemmin kuin pelkkä upotusten samankaltaisuus. LangChainin CrossEncoderReranker-komponentti yhdessä ContextualCompressionRetriever-komponentin kanssa toteutti tämän: ensin perushakija (esim. hybridi- tai monikyselyhakija) hakee laajemman joukon dokumentteja, jonka jälkeen cross-encoder-malli arvioi ne ja palauttaa vain parhaiten relevantiksi arvioidut (top_n) dokumentit jatkokäsittelyyn (Esimerkkikoodi 3). Tämä vaihe on kriittinen, koska se parantaa LLM:lle syötettävän kontekstin laatua ja vähentää ns. kohinaa.

```
# Load the cross-encoder model locally
reranker_model = HuggingFaceCrossEncoder(
     model_name="BAAI/bge-reranker-v2-m3"
)

# Initialize the re-ranker component
compressor = CrossEncoderReranker(
     model=reranker_model,
     top_n=15  # Number of documents to keep after re-ranking
)

compression_retriever = ContextualCompressionRetriever(
     base_compressor=compressor,
     base_retriever=retriever
)

# Helper function to perform retrieval and re-ranking
def retrieve_and_rerank(query):
     compressed_docs = compression_retriever.invoke(query)
     context = "\n\n".join(doc.page_content for doc in compressed_docs)
      return {"context": context, "question": query}

# [...] Code for RAG chain setup [...]
```

Esimerkkikoodi 3. Uudelleenjärjestäjän (Re-ranker) konfigurointi. Koodi alustaa CrossEncoderReranker-komponentin (compressor) käyttäen paikallisesti ladattua BAAI/bge-reranker-v2-m3 cross-encoder-mallia. Tämä kääritään ContextualCompressionRetriever-olioon, joka ensin hakee dokumentit perushakijalla (base_retriever) ja sitten järjestää ja suodattaa ne (top_n=15) uudelleenjärjestäjän avulla ennen niiden palauttamista. - Uudelleenjärjestämisen (Re-ranking) toteutus query_data -funktiossa. HuggingFaceCrossEncoder-malli ladataan ja sitä käyttävä CrossEncoderReranker (compressor) alustetaan palauttamaan top_n parasta dokumenttia. ContextualCompressionRetriever yhdistää tämän monikyselyhakijan (retriever) kanssa. retrieve_and_rerank-apufunktio suorittaa tämän yhdistetyn haun ja uudelleenjärjestämisen.

5.4 Vastausten generointi (Generation)

Kun relevantti konteksti on haettu ja järjestetty uudelleen edellisessä vaiheessa (Tiedonhaku 5.3), se on valmis syötettäväksi kielimallille (LLM) yhdessä käyttäjän alkuperäisen kyselyn kanssa lopullisen vastauksen luomiseksi. Tämä generointivaihe hyödyntää LangChainin ketjutusmekanismia (Chain), joka mahdollistaa monimutkaisten prosessien rakentamisen yhdistämällä eri komponentteja modulaarisesti ja hallitusti.

Vastauksen generointia ohjataan kehotepohjalla (PromptTemplate). Kehotepohja antaa LLM:lle selkeät ohjeet siitä, miten sen tulee hyödyntää annettua,

uudelleenjärjestettyä kontekstia (context) vastatakseen alkuperäiseen kysymykseen (question). Tässä toteutuksessa käytetty kehotepohja (Esimerkkikoodi 4) neuvoo mallia vastaamaan kysymykseen annettujen kontekstitietojen perusteella, olemaan keksimättä tietoa (hallusinoimaan) ja ilmoittamaan selvästi, jos vastausta ei löydy annetusta kontekstista tai jos kysymys ei liity kontekstiin. Tämä on kriittistä luotettavien ja faktoihin perustuvien vastausten varmistamiseksi.

Lopullinen vastausketju (rag_chain) rakennetaan yhdistämällä tiedonhaku- ja uudelleenjärjestämisvaihe (retrieve_and_rerank-funktio, joka palauttaa kontekstin ja kysymyksen), kehotepohja (custom_rag_prompt), itse kielimalli (llm) ja tuloksen jäsentäjä (response_parser, tässä StrOutputParser joka varmistaa tekstimuotoisen tulosteen). LangChainin RunnablePassthrough-komponenttia käytetään ketjun alussa varmistamaan, että alkuperäinen käyttäjän kysely välittyy oikein retrieve_and_rerank-vaiheelle. Putkitusoperaattori (|) yhdistää nämä komponentit saumattomaksi kokonaisuudeksi, joka ottaa vastaan käyttäjän kyselyn ja palauttaa kielimallin generoiman vastauksen (Esimerkkikoodi 4).

```
# [...] Code for retriever and re-ranker setup [...]

    # Define the prompt template for the LLM
    prompt_template = """
    You are a question answering assistant. Use the following pieces
    of context to provide a detailed, accurate, and informative answer
    to the question at the end.
    If you don't know the answer, just say that you don't know, don't
    try to make up an answer.
    Do not answer the question if there is no given context.
    Do not answer the question if it is not related to the context.

    Context:
    {context}

    Question: {question}
    """

    custom_rag_prompt = PromptTemplate.from_template(prompt_template)

    # Define the output parser
    response_parser = StrOutputParser()

    # Construct the final RAG chain
    rag_chain = (
        RunnablePassthrough()   # Passes the initial query through
        | retrieve_and_rerank   # Retrieves and re-ranks documents
        | custom_rag_prompt     # Prompt with context and query
        | llm                   # Formatted prompt to the LLM
        | response_parser       # Parses the LLM output into a string
    )

    # Invoke the chain with the user query
    return rag_chain.invoke(query)
```

Esimerkkikoodi 4. RAG-vastausketjun (rag_chain) rakentaminen ja suoritus query_data -funktion loppuosassa. Ketju alkaa RunnablePassthrough:lla välittäen alkuperäisen kyselyn. retrieve_and_rerank -funktio hakee ja uudelleenjärjestää relevantin kontekstin. custom_rag_prompt muotoilee kehotteen LLM:lle käyttäen saatua kontekstia ja kysymystä. llm (ChatOllama-instanssi) generoi vastauksen tämän kehotteen perusteella. Lopuksi response_parser muuntaa LLM:n vastauksen merkkijonoksi. Koko ketju suoritetaan invoke-metodilla käyttäjän antamalla kyselyllä (query).

Lisäksi järjestelmään toteutettiin valinnainen vastauksen jatkojalostus (refinement). Jos tämä toiminto on käytössä, järjestelmä suorittaa käytännössä toisen RAG-kierroksen: LLM:n generoima ensimmäinen vastaus otetaan talteen, minkä jälkeen suoritetaan uusi tiedonhaku alkuperäisen kyselyn perusteella. Tämän jälkeen ensimmäinen vastaus, alkuperäinen kysely ja uudelleen haettu konteksti syötetään LLM:lle erityisen tarkennuskehotteen (refine_template) kanssa. Tämä kehotepohja ohjeistaa mallia arvioimaan alkuperäistä vastausta

suhteessa kontekstiin ja kysymykseen ja tuottamaan parannellun version, pyrkien korjaamaan epätarkkuuksia, lisäämään puuttuvaa olennaista tietoa ja poistamaan epärelevantteja osia.

5.5 Käyttöliittymä

Jotta RAG-järjestelmän testaaminen ja käyttö olisi mahdollisimman helppoa ja interaktiivista, sille toteutettiin selainpohjainen käyttöliittymä käyttäen Pythonin Streamlit-kirjastoa. Streamlit valittiin sen nopeuden ja helppokäyttöisyyden vuoksi, sillä se mahdollistaa datavetoisten verkkosovellusten rakentamisen suoraan Python-koodilla ilman tarvetta erillisille frontend-kehityksen työkaluille.

Toteutettu käyttöliittymä (Kuva 2) tarjoaa käyttäjälle seuraavat toiminnallisuudet:

1. Chat-ikkuna: Käyttäjä voi kirjoittaa kysymyksensä tekstikenttään ja lähettää sen järjestelmälle. Järjestelmän generoima vastaus ilmestyy chat-historiaan kysymyksen alapuolelle.

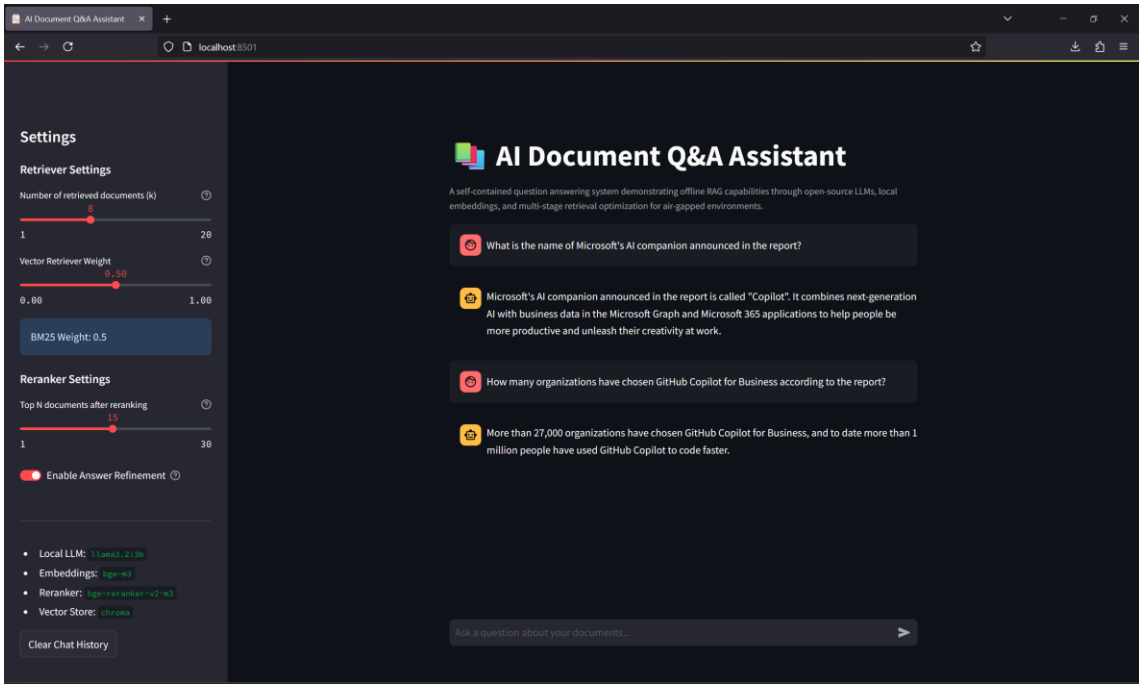
2. Parametrien säätö (Sivupalkki): Käyttöliittymän sivupalkista käyttäjä voi reaaliaikaisesti säätää keskeisiä RAG-prosessin parametreja, mikä mahdollistaa eri asetusten vaikutuksen helpon testaamisen:

    - Haettavien dokumenttien määrä (retriever_k): Kuinka monta dokumenttipalaa hakija (hybridi/monikysely) palauttaa.
    - Hybridihakijan painoarvot: Säätää BM25-avainsanahaun ja semanttisen vektorihaun välistä painotusta hybridihakijassa (arvon ollessa 0.5, molempien paino on yhtä suuri).
    - Uudelleenjärjestäjän top_n (reranker top_n): Kuinka monta dokumenttia uudelleenjärjestäjä palauttaa relevantteina LLM:lle annettavaksi.
    - Vastauksen jatkojalostus (Enable Answer Refinement): Valintakytkin, jolla voi ottaa käyttöön tai poistaa käytöstä luvussa 5.4 kuvatun vastauksen tarkentamisen.

3. Keskusteluhistorian tyhjennys: Painike, jolla voi tyhjentää näytetyn keskusteluhistorian ja siihen liittyvän sovelluksen muistin (Streamlit session state), mahdollistaen uuden testauksen aloittamisen puhtaalta pöydältä.

4. Informaatio: Käyttöliittymä näyttää tietoa käytössä olevista paikallisista malleista (LLM, Embeddings, Reranker) sekä käytössä olevan vektoritietokannan.

Tämä interaktiivinen käyttöliittymä tekee järjestelmän toiminnan tutkimisesta, eri parametrien kokeilemisesta ja tulosten arvioinnista helppoa ja havainnollista.



Kuva 2. Streamlit-pohjainen käyttöliittymä paikalliselle RAG-järjestelmälle. Käyttöliittymä sisältää chat-ikkunan kysymysten esittämiseen ja vastausten tarkasteluun sekä sivupalkin, josta voidaan säätää RAG-prosessin parametreja, kuten haettavien dokumenttien määrää, hybridihakijan painotusta, uudelleenjärjestämisen laajuutta ja vastauksen jatkojalostuksen aktivointia.

5.6 Testaus ja arviointi

Tässä alaluvussa kuvataan toteutetun paikallisen RAG-järjestelmän toiminnallisuuden testausta ja arviointia. Testauksen tavoitteena oli saada käsitys järjestelmän kyvystä vastata kysymyksiin annetun dokumenttiaineiston perusteella sekä arvioida tiettyjen kehittyneiden komponenttien, kuten hybridihakutekniikan ja valinnaisen vastausten jatkojalostuksen (refinement), vaikutusta vastausten laatuun ja relevanssiin.

5.6.1 Testausmenetelmä

Testaus suoritettiin käyttämällä kahta valittua PDF-muotoista dokumenttia testidatana. Dokumenteille muodostettiin ennalta määritelty joukko testikysymyksiä, jotka kattoivat erilaisia faktatietoja, kuten nimiä, määriä ja keskeisiä avainlukuja. Oikeat vastaukset kysymyksiin varmistettiin etukäteen testidokumenteista.

Järjestelmän tuottamia vastauksia arvioitiin kvalitatiivisesti ihmisarvioinnin keinoin. Arvioinnissa kiinnitettiin huomiota vastauksen:

1. Oikeellisuuteen: Vastasiko vastaus kysymykseen ja oliko se faktuaalisesti oikein dokumentin tietojen perusteella?

2. Relevanssiin: Liittyikö vastaus suoraan kysymykseen?

3. Kattavuuteen: Sisälsikö vastaus olennaisen tiedon, vai oliko se liian suppea tai liian laaja?

Erityisesti jatkojalostustoiminnon (refinement) testaamiseksi järjestelmän ensimmäinen, ilman jatkojalostusta generoitu vastaus kirjattiin lokitietoihin, ja lopullista vastausta verrattiin sekä alkuperäiseen vastaukseen että dokumentista varmistettuun oikeaan tietoon.

5.6.2 Testauksen havainnot

Testauksen aikana tehtiin seuraavia keskeisiä havaintoja:

1. Perus-RAG ja Hybridihaku: Järjestelmä kykeni pääosin löytämään relevantteja asiayhteyksiä ja generoimaan vastauksia annettuihin kysymyksiin paikallisten dokumenttien pohjalta. Hybridihaku (semanttinen + BM25) vaikutti parantavan relevanssia verrattuna pelkkään semanttiseen hakuun, erityisesti kysymyksissä, jotka sisälsivät tarkkoja termejä tai nimiä.

2. Uudelleenjärjestäminen (Re-ranking): Cross-encoder-pohjainen uudelleenjärjestäjä näytti priorisoivan hakutuloksia tehokkaasti, mikä todennäköisesti paransi LLM:lle syötettävän kontekstin laatua. Vaikka tämä on yleinen oletus, tämän vaikutuksen laajempaa konkreettista testausta ei tässä työssä tehty.

3. Jatkojalostus (Refinement):

    a. Jatkojalostus onnistui joissakin tapauksissa täydentämään vastausta kontekstista löytyvällä relevantilla lisätiedolla, tehden vastauksesta kattavamman.

    b. Sen kyky korjata merkittävästi virheellisiä alkuperäisiä vastauksia oli kuitenkin rajallinen. Useimmiten, jos alkuperäinen vastaus oli selvästi väärä, myös jatkojalostettu vastaus jäi epätarkaksi tai virheelliseksi.

    c. Parhaiten korjaus toimi numeeristen tietojen kohdalla, joissa pieniä epätarkkuuksia saatettiin korjata toisella kierroksella.

    d. Jatkojalostus pidensi merkittävästi vastauksen saamisen viivettä ja lisäsi laskennallista kuormaa odotetusti.

5.6.3 Testauksen rajoitukset

On tärkeää huomioida, että tämän testauksen laajuus oli rajallinen. Se perustui valittuihin testidokumentteihin ja rajattuun määrään kysymyksiä. Arviointi oli kvalitatiivista, eikä systemaattisia kvantitatiivisia mittareita (kuten tarkkuus, F1-score) käytetty. Tulokset antavat suuntaa järjestelmän toiminnasta, mutta laajemmat johtopäätökset vaatisivat kattavampaa testausta erilaisilla aineistoilla ja kyselytyypeillä.

6 Yhteenveto

Tämän insinöörityön tavoitteena oli tutkia ja kehittää paikallisessa käyttöympäristössä toimiva, tekoälypohjainen kysymys-vastaussovellus (Q&A). Työ keskittyi hyödyntämään avoimen lähdekoodin suuria kielimalleja (LLM) ja moderneja Retrieval-Augmented Generation (RAG) -tekniikoita ilman riippuvuutta ulkoisista pilvipalveluista. Päätavoitteena oli luoda ratkaisu, joka pystyy vastaamaan käyttäjän kysymyksiin paikallisesti tallennettujen dokumenttiaineistojen perusteella, parantaen näin tietosuojaa ja hallittavuutta.

Työssä tehdyn järjestelmän toteutus perustui LangChain-ohjemistokehykseen, joka mahdollisti modulaarisen rakenteen. Keskeisenä osana paikallista suoritusta hyödynnettiin Ollama-sovellusta, joka mahdollisti avoimen lähdekoodin LLM-mallien (Llama 3.2) ja upotusmallien (bge-m3) lataamisen ja ajamisen paikallisesti. Dokumenttien esikäsittely ja indeksointi toteutettiin vaiheittaisena prosessina Chroma-vektoritietokantaan. Tiedonhaussa sovellettiin kehittyneitä menetelmiä, kuten hybridihakua, monikyselyhakua ja cross-encoder-pohjaista uudelleenjärjestämistä, parantamaan hakutulosten laatua ennen niiden syöttämistä LLM:lle. Interaktiivinen käyttöliittymä toteutettiin Streamlit-kirjastolla.

Työn tavoite paikallisesti toimivan, avoimen lähdekoodin LLM:ää ja RAG-tekniikkaa hyödyntävän Q&A-sovelluksen kehittämisestä saavutettiin. Toteutettu RAG-järjestelmä kykeni tuottamaan relevantteja ja kontekstisidonnaisia vastauksia käyttäjän kysymyksiin paikallisesti tallennetuista dokumenteista ilman pilvipalveluita, parantaen näin tietosuojaa ja hallittavuutta. Keskeinen havainto oli, että kehittyneiden hakutekniikoiden, kuten hybridihaku (semanttinen + BM25), soveltaminen paransi hakutulosten tarkkuutta ja kattavuutta verrattuna perustason semanttiseen hakuun.

Keskeiseksi selvittämättä jääneeksi osa-alueeksi muodostui kuitenkin monimuotoisen datan, kuten taulukoiden ja kuvien, tehokas käsittely. Vaikka perustason tekstinkäsittely onnistui, nykyisen järjestelmän kyky hyödyntää ei-tekstuaalista tietoa on rajallinen. Tämä osoittaa, että RAG-järjestelmien toteutus ei ole

suoraviivaista ja niiden suorituskyky on vahvasti riippuvainen käsiteltävän aineiston luonteesta ja muodosta, eikä yhtä kaikille sopivaa ratkaisua ole.

Tämä insinöörityö tarjoaa päätelmänään, että paikallisesti toimivat RAG-järjestelmät, jotka hyödyntävät jopa suhteellisen pieniä avoimen lähdekoodin kielimalleja, ovat merkittävä ja käyttökelpoinen ratkaisu organisaatioille. Ilman tätä työtä konkreettinen osoitus avoimen lähdekoodin kielimallien käytöstä ja näiden tekniikoiden (hybridihaku, monikysely) vaikutuksesta paikallisen RAG-järjestelmän suorituskykyyn ja toteutettavuuteen voisi olla vähäisempää. Työn tuloksia voidaan hyödyntää esimerkiksi organisaation sisäisessä tiedonhaussa, asiakaspalvelun tukena tai muissa tietoturvaa ja datan hallintaa vaativissa sovelluksissa. Saadut tulokset antavat pohjan jatkotutkimukselle RAG-järjestelmien suorituskyvyn optimoimiseksi ja skaalautuvuuden parantamiseksi paikallisissa ympäristöissä.

Tunnistettujen haasteiden ja järjestelmän potentiaalin pohjalta jatkotutkimuksessa voitaisiin keskittyä erityisesti seuraaviin osa-alueisiin:

1. Monimuotoisen datan käsittely:

    a. Taulukoiden tunnistus ja muunto: Kehittyneemmät menetelmät taulukoiden automaattiseksi tunnistamiseksi eri dokumenttimuodoista ja niiden luotettavaksi muuntamiseksi LLM:lle sopivaan muotoon (esim. Markdown).

    b. OCR- ja kuvankäsittelytuki: Tekstintunnistuksen (OCR) integrointi skannattujen dokumenttien ja kuvapohjaisten PDF-tiedostojen sisällön hyödyntämiseksi, esimerkiksi avoimen lähdekoodin OCR-kirjastojen (kuten Tesseract) avulla tai integroimalla OCR-kyvykkyyden sisältäviä malleja esikäsittelyvaiheeseen.

    c. Multimodaalisten ja visuaalisten mallien käyttö: Visuaalisten kielimallien (VLM), kuten Meta Llama 3.2 Vision tai Microsoft Phi-4-Multimodal [31], tutkiminen ja integrointi. Nämä mallit

mahdollistavat PDF-tiedostojen kuvasivujen ja upotettujen diagrammien suoran analysoinnin yhdessä tekstisisällön kanssa ilman erillistä OCR-vaihetta. Tämä vähentää virheitä, parantaa kontekstin ymmärtämistä ja tekee tiedonhausta kokonaisvaltaisempaa. Vaikka VLM:t vaativat tyypillisesti enemmän laskentatehoa, mallien kvantisointi- ja optimointimenetelmät mahdollistavat niiden käytön myös resurssirajoitteisemmissa ympäristöissä, kun taas suuremmat versiot vaativat tehokkaampaa GPU-laskentaa.

2. Hakumenetelmien ja vastausgeneroinnin optimointi:

    a. Hybridihakijan dynaaminen painotus: Mekanismien kehittäminen, jotka säätävät semanttisen ja avainsanahaun painoarvoja dynaamisesti kyselyn luonteen perusteella.

    b. Parametrien automaattinen optimointi: Haku- ja generointiprosessien parametrien (esim. palakoko, top_k, top_n, temperature) automaattinen säätö perustuen dataan tai käyttäjäpalautteeseen.

    c. Konteksti-ikkunan hallinta: Tehokkaampien strategioiden tutkiminen kontekstin tiivistämiseksi ja uudelleenjärjestämiseksi LLM:n rajoitetun konteksti-ikkunan puitteissa.

    d. Kyselynlaajennustekniikoiden vertailu: Eri tapojen tutkiminen kyselyiden monimuotoistamiseksi (multi-query) ja niiden vaikutuksen arviointi hakutuloksiin.

Yhteenvetona voidaan todeta, että paikallisesti toimivilla RAG-järjestelmillä on merkittävä potentiaali tietoturvallisen ja luotettavan tiedonhaun toteuttamiseen ilman pilviriippuvuutta. Vaikka kehittäminen on monivaiheista ja sisältää haasteita, erityisesti datan monimuotoisuuden osalta, jatkokehitys voi tarjota entistä tehokkaampia työkaluja paikalliseen tiedonhakuun monimutkaisissa ja asiantuntijavetoisissa käyttöympäristöissä.

Lähteet

1 Kerner, Sean Michael. 2025. DeepSeek explained: Everything you need to know. Verkkoaineisto. TechTarget. <https://www.techtarget.com/whatis/feature/DeepSeek-explained-Everything-you-need-to-know>. 6.2.2025.

2 Open LLMs for Transparent AI in Europe. 2025. Verkkoaineisto. OpenEuroLLM. <https://openeurollm.eu/launch-press-release>.

3 University of Turku partner in major consortium to advance European AI capacity. 2025. Verkkoaineisto. University of Turku. <https://www.utu.fi/en/news/news/university-of-turku-partner-in-major-consortium-to-advance-european-ai-capacity>. 5.2.2025.

4 Swain, Gyana. 2025. EU supports AI challenge to Silicon Valley and China. Verkkoaineisto. InfoWorld. <https://www.infoworld.com/article/3815484/eu-supports-ai-challenge-to-silicon-valley-and-china.html>. 3.2.2025.

5 OpenEuroLLM: a European open source AI language models project. 2025. Verkkoaineisto. datos.gob.es. <https://datos.gob.es/en/noticia/openeurollm-european-open-source-ai-language-models-project>. 11.3.2025.

6 Robison, Kylie. 2024. OpenAI cofounder Ilya Sutskever says the way AI is built is about to change. Verkkoaineisto. <https://www.theverge.com/2024/12/13/24320811/what-ilya-sutskever-sees-openai-model-data-training>. 14.12.2024.

7 Transformers. Verkkoaineisto. Hugging Face. <https://huggingface.co/docs/transformers/>.

8 Attention is All You Need. 2017. Verkkoaineisto. arXiv. <https://arxiv.org/abs/1706.03762>. Päivitetty 2.8.2023.

9 Attention is All You Need. Verkkoaineisto. Google. <https://research.google/pubs/attention-is-all-you-need/>.

10 What is OpenAI’s GPT series? Verkkoaineisto. Milvus. <https://milvus.io/ai-quick-reference/what-is-openais-gpt-series>.

11 ChatGPT. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/ChatGPT>. Päivitetty 8.4.2025.

12 Heaven, Will Douglas. 2023. Rogue superintelligence and merging with machines: Inside the mind of OpenAI’s chief scientist. Verkkoaineisto. MIT Technology Review. <https://www.technologyreview.com/2023/10/26/1082398/exclusive-ilya-sutskever-openais-chief-scientist-on-his-hopes-and-fears-for-the-future-of-ai/>. 26.10.2023.

13 Carter, Tom. 2023. We didn't think ChatGPT was very good when we released it, OpenAI's chief scientist says. Verkkoaineisto. Business Insider. <https://www.businessinsider.com/chatgpt-was-inaccurate-boring-when-it-launched-openai-cofounder-2023-10>. 30.10.2023.

14 Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. 2020. Verkkoaineisto. arXiv. <https://arxiv.org/abs/2005.11401>. Päivitetty 24.2.2021.

15 Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Verkkoaineisto. Meta. <https://research.facebook.com/publications/retrieval-augmented-generation-for-knowledge-intensive-nlp-tasks/>.

16 Tiwari, Juhi. What is RAG - Retrieval-Augmented Generation? Verkkoaineisto. Kore.ai. <https://blog.kore.ai/understanding-retreival-augmented-generation-rag>.

17 LlamaIndex High-Level Concepts. Verkkoaineisto. LlamaIndex. <https://docs.llamaindex.ai/en/stable/getting_started/concepts/>.

18 Ollama API. Verkkoaineisto. GitHub. <https://github.com/ollama/ollama/blob/main/docs/api.md>.

19 Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. 2024. Verkkoaineisto. Meta AI. <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>. 25.9.2024.

20 bge-m3. Verkkoaineisto. Ollama Library. <https://ollama.com/library/bge-m3>.

21 BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. 2024. Verkkoaineisto. arXiv. <https://arxiv.org/abs/2402.03216>. Päivitetty 28.6.2024.

22 LangChain Introduction. Verkkoaineisto. LangChain. <https://python.langchain.com/docs/introduction/>.

23 Chroma is the open-source AI application database. Batteries included. Verkkoaineisto. Chroma. <https://www.trychroma.com/>.

24 Streamlit. A faster way to build and share data apps. Verkkoaineisto. Streamlit. <https://streamlit.io/>.

25 Optimizing RAG with Hybrid Search & Reranking. 2024. Verkkoaineisto. VectorHub by Superlinked. <https://superlinked.com/vectorhub/articles/optimizing-rag-with-hybrid-search-reranking>. 18.3.2024.

26 How to combine results from multiple retrievers. Verkkoaineisto. LangChain. <https://python.langchain.com/docs/how_to/ensemble_retriever/>.

27 RAG-Fusion: Multi-Query Retrieval & Rank Fusion Techniques. 2024. Verkkoaineisto. DocsBot. <https://docsbot.ai/article/advanced-rag-techniques-multiquery-and-rank-fusion>. 30.3.2024.

28 How to use the MultiQueryRetriever. Verkkoaineisto. LangChain. <https://python.langchain.com/docs/how_to/MultiQueryRetriever/>.

29 Ism, Ilias. 2024. Reranking Explained: Why It Matters for RAG Systems. Verkkoaineisto. Chatbase. <https://www.chatbase.co/blog/reranking>. 29.9.2024.

30 BAAI/bge-reranker-v2-m3. Verkkoaineisto. Hugging Face. <https://huggingface.co/BAAI/bge-reranker-v2-m3>.

31 kinfey. 2025. Welcome to the new Phi-4 models - Microsoft Phi-4-mini & Phi-4-multimodal. Verkkoaineisto. Microsoft Tech Community. <https://techcommunity.microsoft.com/blog/educatordeveloperblog/welcome-to-the-new-phi-4-models---microsoft-phi-4-mini--phi-4-multimodal/4386037>. 27.2.2025.