ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Bihami Bence Kristóf**
2025	Hallgató törzskönyvi száma:	**T/008525/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Biomatika és Alkalmazott Mesterséges Intelligencia Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve:	**Bihami Bence Kristóf**
Törzskönyvi száma:	T/008525/EI12904/N
Neptun kódja:	[redacted]

A dolgozat címe:

Tartalom-összefoglaló MI
Content Summarizer AI

Intézményi konzulensek:	Dr. Eigner György, Szigeti Mark Bence
Külső konzulens:	
Beadási határidő:	2024. december 15.
A záróvizsga tárgyai:	Számítógép architektúrák Mesterséges intelligencia specializáció

A feladat

A digitális korban az emberiség tudása és információszerzési módszerei merőben digitalizálódott irányba haladtak. Nap mint nap hatalmas mennyiségű digitális dokumentum, képanyag és hangfelvétel születik, melyek értelmezése és rendszerezése nem kis kihívást jelent.
E jelen szakdolgozat egy olyan mesterséges intelligencia alapú megoldást kíván bemutatni, amely az online térben uralkodó információáradattal képes hatékonyan megbirkózni. Az eszköz képes szöveges, képi és hang alapú tartalmak feldolgozására, mely formátumok a modern digitális kommunikáció alappilléreit képezik. A rendszer az érkező információkat azonnal elemez és egy tömör, könnyen átlátható összefoglalót kínál, melyen keresztül a felhasználó egy virtuális asszisztens segítségével mélyebben eligazodhat az adott témában.
A szakdolgozat elsődleges célja, hogy a felhasználóknak támogatást nyújtson az összetettebb nyelvezetű tartalmak értelmezésében, miközben lehetőséget biztosít arra, hogy az információ teljes körű átnézése nélkül is a lényegre tapintanak.

A dolgozatnak tartalmaznia kell:

- információ extrakciós eljárások bemutatása,
- Text-to-Speech módszerek ismertetése és összevetése,
- nyelvmodell kiválasztásának folyamata,
- MI Ágensek ismertetése és szerepük a feladatban,
- Langchain keretrendszer ismertetése,
- Docker konténerizáció,
- a rendszer implentációja,
- továbbfejlesztési lehetőségek ismertetése.

ÓBUDAI EGYETEM Neumann János Informatikai Kar 1. Budapest

Ph.

.....
Sájevicsné Dr. habil. Sápi Johanna
intézetigazgató

A szakdolgozat elévülésének határideje: **2026. december 15.**
(ÓE HKR 54.§ (10) bekezdés szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2024.12.15. .....

Bihami Bence

.....

hallgató aláírása

ABSZTRAKT

A szakdolgozatom központi témája a szöveges és médiaalapú tartalmak minél pontosabb összefoglalása mesterséges intelligencia segítségével. Az elemzés egy kezdeti problémából indul ki, amelyet irodalomkutatási módszerekkel alaposan körüljárok. Célom, hogy a témához kapcsolódó legfontosabb fogalmakat, szakmai kifejezéseket és módszereket rendszerezetten és nagy precizitással összegyűjtsem, előkészítve ezzel a gyakorlati megvalósítást.

Az elméleti háttér feldolgozását követően kidolgozok egy megoldási javaslatot, amely a tudományos alapelvek és a gyakorlati alkalmazás közötti kapcsolatot helyezi előtérbe. Ennek részeként egy részletes rendszertervet készítek, amely meghatározza a szoftverfejlesztési folyamat irányvonalát.

A rendszertervre támaszkodva egy mesterséges intelligencia-alapú alkalmazást hozok létre, amely a választott probléma megoldására szolgál. A fejlesztési folyamat során dokumentálom a lépéseket és a felmerülő kihívásokat, részletezve a tapasztalatokat.

A szakdolgozat lezárásaként javaslatokat fogalmazok meg a szoftver további fejlesztési lehetőségeire, megnyitva ezzel az utat a jövőbeni kutatások és innovációk előtt.

ABSTRACT

The central topic of my thesis is the most accurate possible summarization of textual and media-based content using artificial intelligence. The analysis begins with an initial problem, which I thoroughly explore using literature research methods. My goal is to systematically and precisely collect the most important concepts, professional terms, and methods related to the topic, thereby preparing for practical implementation.

Following the theoretical background review, I will develop a proposed solution that emphasizes the connection between scientific principles and practical application. As part of this, I will create a detailed system design that defines the direction of the software development process.

Based on the system design, I will create an artificial intelligence-based application designed to solve the chosen problem. During the development process, I will document the steps and challenges encountered, detailing the experiences.

As the conclusion of the thesis, I will formulate suggestions for further development opportunities of the software, thus paving the way for future research and innovations.

TARTALOMJEGYZÉK

1 BEVEZETÉS

A probléma felvezetése és fontossága

A modern információrobbanás korszakában a rendkívüli mennyiségű dokumentum és a szakmai kifejezések használata jelentős kihívást állít az emberek elé. A dokumentumok bősége miatt az átfogó megértésért hosszas időt kellene eltöltenünk interpretációra, ami sokszor frusztráló és kimerítő folyamatot eredményez. Az esetleges értelmezési nehézségek esetén az olvasó gyakran saját magára van utalva, és további kutatásra kényszerül. Az dokumentumokban felfedezhető információk hitelességének ellenőrzése gyakran kihívást jelent, és az olvasónak kritikus gondolkodásra és értékelési készségekre van szüksége ahhoz, hogy megkülönböztesse a megbízható tényeket a megtévesztő vagy pontatlan információktól.

Az olvasóknak hatékony eszközökre és stratégiákra van szükségük a nagy mennyiségű és gyakran bonyolult információ kezeléséhez.

A probléma aktualitása

A vizsgált probléma relevanciája napjainkban folyamatosan növekszik, mivel a számítógépek és a mesterséges intelligencia térnyerése az adatok mennyiségének exponenciális növekedéséhez vezetett. Ez jelentősen akadályozza az embereket abban, hogy lépést tartsanak az információáramlással és hatékonyan feldolgozzák a rendelkezésre álló adatokat. Emellett a strukturált adatok értéke az elmúlt években robbanásszerűen megnőtt, ami tovább fokozza az igényt olyan programok iránt, amelyek képesek ismeretlen tartalmú állományokból automatikusan kulcsszavakat kinyerni vagy tartalmi összefoglalókat készíteni. Ilyen megoldások iránt jelenleg rendkívül nagy a kereslet, különösen az adatkezelési és információs rendszerek területén.

A szakdolgozat célja

A szakdolgozat célja egy hatékony és korszerű megoldás bemutatása, amely pontos feltételek mellett képes csökkenteni a terjedelmes szövegek elolvasásához és megértéséhez szükséges időt. A szövegek összefoglalása során kiemelt figyelmet kell fordítani az eredmény olvashatóságára és természetességére, elkerülve a zavaró tényezőket. Az eszköz célja a felhasználó támogatása abban, hogy hosszabb dokumentumokat a lehető leggyorsabban és legalaposabban megértsen, mindezt úgy, hogy a rendszer működése és az emberi munka közötti különbség észrevétlen maradjon.

2 IRODALOMKUTATÁS

2.1 Információ extrakciós eljárások bemutatása

Az elmúlt években megugrott a strukturálatlan adatok mennyisége szövegek, videók, hanganyagok és fényképek formájában. A Natural Language Understanding (NLU) – Természetes Nyelvi Értelmezés képes segíteni a szövegekből való értékes információk kinyerésében, például a közösségi média adatokból, az ügyfelek felméréseiből és a panaszokból. [1]

A Természetes Nyelvfeldolgozás, avagy a Natural Language Processing (NLP)

Az informatikai szektorban már hosszú ideje intenzív erőfeszítések folynak a nyelvfeldolgozás területén. Ahogyan a neve is mutatja a törekvések arra irányulnak, hogy a számítógépek képesek legyenek az írásos dokumentumokat és a beszédet az emberekhez hasonlóan megérteni. Az NLP kombinálja a számítógépes nyelvészetet – az emberi nyelv szabályalapú modellezését – statisztikai, gépi tanulási és mély tanulási modellekkel. Ezek a technológiák együttesen lehetővé teszik a számítógépek számára, hogy az emberi nyelvet szöveg vagy hangadatok formájában dolgozzák fel, és „megértsék” annak teljes jelentését, kiegészítve a beszélő vagy író szándékaival és érzelmeivel. [2]

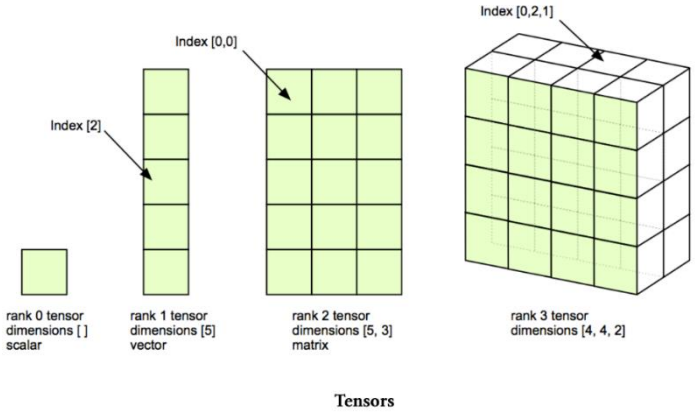
Az NLP önmagában arra törekszik, hogy a strukturálatlan nyelvi adatokat strukturált adatformátummá alakítsa át. Az értelmes kommunikációhoz két komponensére van szüksége melyek közül az NLU, a nyelvtanon és a szövegkörnyezeten keresztül az emberi nyelv struktúrájának és jelentésének mélyreható megértését teszi lehetővé átalakítva gépi olvasásértésre, így lehetővé téve a mondat szándékolt jelentésének meghatározását. A másik alkotó egysége a Natural Language Generation (NLG) - Természetes Nyelvi Generálás, amely nagy hangsúlyt fektet az adathalmazából történő szöveg szilánkok összefüggő értelmes kimenetként való egyesítésre angol vagy más nyelven való átalakítással. [3]

A nyelvfeldolgozás fejlődése során számos kihívással találkozott, beleértve a különböző nyelvek közötti eltéréseket és az azonos nyelvet beszélő emberek regionális nyelvi variációit. Az NLP technológiai ág feladata megbirkózni a folyamatosan fejlődő próbatételek széles skálájával, csökkentve a távolságot a gépi és a természetes értelmezés között.

A nyelvfeldolgozás adatstrukturáló eszközei a következőek.

A **tensors** (tenzorok) alapvető adatstruktúrák a matematikában és a fizikában, amelyek skalárokat, vektorokat és mátrixokat általánosítanak. Ezek többdimenziós tömbök, amelyek képesek nagy mennyiségű numerikus adat tárolására és hatékony kezelésére. A tenzorok lehetővé teszik a szöveges adatok megjelenítését és manipulálását szavak,

mondatok vagy dokumentumok numerikus vektorokként történő kódolásával. Ez a numerikus ábrázolás lehetővé teszi a mély tanulási modellek számára a szöveges adatok hatékony feldolgozását és tanulását. [4]

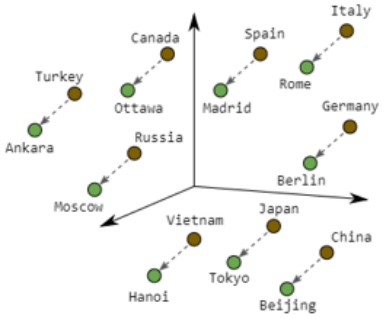


2.1. ábra: Tenzor bemutatása vizuálisan [62]

Az **embeddings** (szóbeágyazás) a kifejezések sűrű vektoros ábrázolásai folytonos vektortérben. Céljuk a szavak numerikus vektorokként való megjelenítéssel szemantikai hasonlóságokat és különbségeket rögzíteni numerikus értékekként, lehetővé téve az algoritmusok számára, hogy a szavakkal megegyező információ készlettel dolgozzanak tovább, azonban már megváltozott típusú könnyebben feldolgozható bemenettel. [4]

A természetes nyelvfeldolgozást megvalósító technológiák a következőek.

2.1.1 A Named Entity Recognition (NER) - Entitás felismerés



2.2. ábra: Szavak vektoros ábrázolása [4]

A Named Entity Recognition (NER) a Natural Language Processing (NLP) információkinyerésének egy részfeladata, amely előre meghatározott kategóriákba sorolja a megnevezett entitásokat, például személynevek, szervezetek, helyek, orvosi kódok, időkifejezések, mennyiségek, pénzbeli értékek stb. Az NLP területén ezeknek az entitásoknak a megértése kulcsfontosságú számos alkalmazás számára, mivel gyakran ezek tartalmazzák a legjelentősebb információkat a szövegben. [5]

A Named Entity Recognition (NER) hídként szolgál a strukturálatlan szöveg és a strukturált adatok között, lehetővé téve a gépek számára, hogy átfésüljék a hatalmas strukturálatlan szöveget, és meghatározott darabokat nevesített entitásként azonosítsanak, majd előre meghatározott kategóriákba sorolják őket. [5]

A NER bonyolultsága végett több lépésre bontható.

Tokenizálás, az entitások azonosítása előtt a szövegsorozatot kisebb részekre bontja, úgynevezett tokenekre alakítja át a könnyebb gépi elemzés érdekében, segítve a gépeket az emberi nyelv megértésében. A „szöveg szilánkok" lehetnek szavak, kifejezések vagy akár mondatok. [5]

Az entitás azonosítása történhet különféle nyelvi szabályok vagy statisztikai módszerek segítségével, melyekkel a potenciális megnevezett entitások észlelhetők. Ez magában foglalja a minták felismerését, például a nagybetűket a nevekben vagy bizonyos formátumokat (például dátumokat). Az entitás besorolás folyamata során az entitások azonosítása után előre meghatározott osztályokba sorolják őket, például "Személy", "Szervezet" vagy "Helyszín". Ezt gyakran címkézett adatkészleteken betanított gépi tanulási modellekkel érik el. [5]

Kontextuális elemzés keretében a NER-rendszerek gyakran figyelembe veszik a körülöttük lévő környezetet a pontosság javítása érdekében. Például a „Szikra támadt a fejében" mondatban a szövegkörnyezet segít a rendszernek felismerni a „szikra" -t hirtelen gondolatként, nem pedig két tárgy hirtelen és erőteljesen összeütközéséből származó fény és hőképződés. [5]

Utófeldolgozás fázisában a kezdeti besorolás után utófeldolgozást lehet alkalmazni az eredmények finomításához. Ez magában foglalhatja a kétértelműségek feloldását, a több token entitás összevonását vagy tudásbázisok használatát az entitásadatok javítására. [5]

Különféle entitásfelismerési módszerek állnak rendelkezésre a széles körű szakirodalomban, ebben a kontextusban a legklasszikusabb megközelítéseket ismertetem.

A **Szabály alapú módszerek** manuálisan kialakított szabályokon alapulnak. Nyelvi minták, reguláris kifejezések vagy szótárak alapján azonosítják és osztályozzák a megnevezett entitásokat. Vannak bizonyos területeken, ahol kifejezetten jobban alkalmazhatóak, itt az entitások jól definiáltak, például a standard orvosi szakkifejezések klinikai feljegyzéseiből kinyerhetők. [5]

A **statisztikai módszerek** esetében, olyan modelleket alkalmaznak, mint a Hidden Markov Model (HMM) - Rejtett Markov Modell vagy a Conditional Random Fields (CRF) - Feltételes Véletlenszerű Mezők. A betanítási adatokból származó valószínűségek alapján megjósolják a megnevezett entitásokat. Ezek a módszerek alkalmasak olyan feladatokra, amelyeknél bőséges címkézett adatkészlet áll rendelkezésükre. Erősségük

abban rejlik, hogy kifejezetten jól általánosítanak, köszönhetően a széleskörű tanító adatok rendelkezésre állásának. [5]

A **gépi tanulási módszerek** feltanított osztályozó modellek által tanulnak a címkézett adatokból, hogy megjósolják a megnevezett entitások típusát. A hátránya, hogy a módszernek jelentős mennyiségű címkézett adatra van szüksége.

A **mély tanulási módszerek** olyan gépi tanulási ágazat, amely mesterséges neurális hálózatokat alkalmaz a bonyolult feladatok megoldására. Az algoritmusok mély, hierarchikus rétegekben dolgoznak, és a rendszer magától tanulja meg a reprezentációkat az adatokból.

2.1.2 Sentiment Analysis - Hangulat elemzés

A hangulatelemzés a digitális szöveg elemzésének egyik fontos folyamata annak megállapítására, hogy az üzenet érzelmi tónusa pozitív, negatív vagy semleges-e. A vállalatok általában nagy mennyiségű szöveges adattal rendelkeznek a felhasználói visszajelzések által, amiket a rendszer képes beolvasni, hogy automatikusan meghatározza a kommentelő hangulatát. [6]

Az elemzés bemenetére általában, valamilyen forrásból előkészített dokumentum kerül, majd a feldolgozás végén egy osztályt (kategóriát, amibe sorolta a szöveget) ad vissza az eljárás kimeneteként.

A **Naive Bayes** a Thomas Bayes által kidolgozott tételén alapuló statisztikai osztályozási technika. Ez az egyik legegyszerűbb felügyelt tanulási algoritmus. A naiv Bayes osztályozók nagy pontossággal és sebességgel rendelkeznek nagy adatkészleteken. Az eljárás azt feltételezi, hogy egy adott tulajdonság hatása egy osztályban független a többi jellemzőtől, még akkor is, ha ezek a jellemzők kölcsönösen függenek egymástól. Ez a feltevés leegyszerűsíti a számítást, ezért tekinthető naivnak. [7]

A hangulatelemzés típusai a következőek.

A **finomszemcsés hangulatelemzés** a hangulatmutatókat pontosabb kategóriákra bontja, mint például a nagyon pozitív és a nagyon negatív. Ez a megközelítés hasonlít az egytől öt csillagig terjedő skálán történő véleményértékeléshez. [8]

Az **érzelemfelismerés**, mint ahogy a neve is mutatja a boldog, szomorú, vidám, dühös stb. affektív funkciók érzelmi csoportokba kerülnek. A hangulatelemzés lexikon módszereként is ismert. [8]

A **szempont alapú elemzés**, azt helyezi magas prioritásba, hogy mi az író nézőpontja. Az, így kialakított kiindulás alapján egy értékkel látja el, amely a hangulat teljes kiértékelésénél egy szempontot jelent. [9]

A **szándék alapú elemzés** a vélemény mellett a szöveg mögötti motivációkat is felismeri. A szövegben található egyes kifejezéses, akár tartalmától függetlenül aktivitásra buzdító háttérjelentéssel is bírhatnak. [8]

A hangulatelemzéssel kihívásai

A hangulatelemzés egy olyan komplex terület, amely számos kihívással néz szembe az értelmezés során. Az egyik jelentős probléma, hogy az egyes érzelmek megítélése függhet a szövegkörnyezettől is, így negatív hatással lehet az eredményre, ha az előbb említett tér, túl szűk, s ezzel kevés információ hiányában kell az algoritmusnak döntenie a pontos osztályról. Továbbá gyengíti a hatékonyságot, azok a szövegben található becsapós affektív állapotok, mint a kétértelmű és a hamis érzelmek. Ezekről nehéz eldönteni, hogy a szöveg fogalmazójának, milyen szándéka volt vele. [8]

2.1.3 Text Summarization – Szövegösszefoglalás

A szövegösszefoglalás az a technika, amellyel tömör és precíz összefoglalót készíthetünk terjedelmes szövegekről, miközben a hasznos információkat közvetítő részekre összpontosítunk anélkül, hogy elveszítené az általános jelentést. [10]

A kivonat alapú összegzés során a szöveg legmeghatározóbb pontjait képviselő szavakat kigyűjtik a szövegből épp úgy, mintha csak a szöveg olvasása közben egy szövegkiemelővel jelölné a fontos információkat, amelyekből továbbiakban meghatározza a szöveg tartalmát rövidebb formában. [10]

Különféle algoritmusok és módszerek alkalmazhatók a mondatok súlyának értékelésére, majd ezek relevanciájuk és egymáshoz való hasonlóságuk alapján történő rangsorolására. Az eredmények összekapcsolhatók egy tartalom összefoglaló program létrehozásához.

2.1.4 Aspect-Based Opinion Mining - Szempont-Alapú Véleménybányászat

A szempont-alapú véleménybányászat magában foglalja az entitás szempontjainak vagy jellemzőinek kinyerését, és az ezekről a szempontokról alkotott véleményeket. Ez egy olyan szövegosztályozási módszer, amely a hangulatelemzésből és az elnevezett entitások kivonásából (NER) fejlődött ki. Az ABOM tehát az aspektuskinyerés és a véleménybányászat kombinációja. [11]

2.1.5 Topic Modeling - Téma modellezés

A témamodellezés az összetettebb módszerek közé sorolható a szövegben helyet foglaló témák azonosítására. A témamodellezés legfőbb előnye, hogy nem felügyelt technika, tehát nincs szükség a modell betanításához hatalmas mennyiségű címkézett adatra. A téma megállapításának problémájára az egyik legnépszerűbb megoldást a Latent Dirichlet Analyse (LDA) - Látens Dirichlet Analízis jelenti. Ez a technika bemenetnek egy dokumentumot vár és egy várható téma kinyerési számot. A folyamat során elemzi a mondatokat és a benne foglalt kifejezésekhez egy fogalom készletet rendel, mely elemei közötti értelmi eltérés alapján feldolgozza a szöveg egyes témáit. [1]

2.2 Text-to-Speech módszerek ismertetése és összevetése

A Text-To-Speech (TTS) technológia írásos adatokat alakít hanggá, és gyakran „hangos olvasó megoldásként” is ismert. Eredetileg a látássérültek és olvasási nehézségekkel küzdők támogatására fejlesztették, de mára széles körben elterjedt, mivel gyors és kényelmes információfeldolgozást tesz lehetővé.

2.2.1 A szöveg beszéddé alakításának folyamata

Szöveg előfeldolgozás

A szöveg előfeldolgozás során a kifejezéseket két csoportra osztják: **szabványos és nem szabványos elemekre**. A szabványos kifejezések általában egyszerűen feldolgozhatók, míg a nem szabványos elemek, például kétértelmű szavak, rövidítések vagy speciális karakterek, mélyebb elemzést igényelnek a helyes kiejtés érdekében. Az előfeldolgozás célja, hogy az algoritmus minden nem szabványos kifejezést, például dátumokat, számokat és egységeket, szabványos szöveggé alakítson. [12]

Fonetikus átírás

Az előfeldolgozás által kifejtett esetek alátámasztják, hogy jobban meg kell határozni a bemeneti szöveg tervezett kiejtését. Ezt általában a fonetikus átírással érik el, egy olyan folyamattal, amely a grafémasorozatot (egy nyelv írásrendszerének legkisebb és szétválaszthatatlan elemének sorozatát) fonémák (a hangzó közlésfolyamat legkisebb megkülönböztethető egységének) sorozatává írja át. [12] [13] [14]

Prozódia elemzés

A szövegnormalizálás és fonetikus átírás után a rendszer figyelembe veheti a beszéd jellemzőit, például a hangszínt, amely életkort vagy regionális akcentust tükröz. Az élvezetes felolvasás érdekében kihívást jelenthet a ritmus, hangsúly és érzelemkifejezés, valamint a hangjelzések, mint a nevetés vagy sóhaj. [12]

A beszélő kódolása

Ez a modell képes kinyerni a beszélő azonosítóját a beszédjelből, ami hasznos lehet több beszélős TTS rendszerekben, ahol a rendszernek képesnek kell lennie arra, hogy különböző hangokat generáljon.

Hangkódoló modell

A Vocoder modellek mély neurális hálózatok, amelyek a spektrogramokból nyers hanghullámokat generálnak, így alakítva a text-to-speech rendszerek kimenetét hallható beszéddé. Ez kulcsfontosságú a természetes hangzású beszéd létrehozásában, mivel a végső hanghullámot érzékeljük és értjük beszédként.

2.2.2 Text-To-Speech módszerek összevetése

A szöveg-beszéd rendszerek két fő módszert alkalmaznak: a **korpusz-alapú** és a **gépi tanulás alapú** eljárásokat.

A korpusz-alapú eljárások előre rögzített beszédadatbázisokból válogatva állítanak össze mondatokat, optimalizálva a beszédminőséget, de korlátozott hanglejtési finomságokkal.

Ezzel szemben a gépi tanulás alapú módszerek a beszéd élethűségének növelésére fókuszálnak, dinamikusan alakítva a hangadatokat. Ez a technológia a TTS rendszerek új generációját képviseli, közelítve a mesterséges beszédet a természetes nyelvhez.

2.2.3 Nagyvállalatok által kifejlesztett text-to-speech modellek

Coqui TTS

A Coqui TTS egy gépi tanuláson alapuló könyvtár szöveg beszéddé alakítására, amely egyszerű használatot, gyors működést és magas beszédminőséget kínál. Előre betanított modellekkel és eszközökkel támogatja az adatkészletek minőségének értékelését, és már több mint 20 nyelven alkalmazzák. A kódoló-dekódoló architektúrán alapuló rendszer API-n keresztül könnyen integrálható, kreditalapú díjazással. [15]

GTTS (Google Text to Speech)

A GTTS egy Python könyvtár, amely a Google fordító motorjával működik és segítségével szövegből képes beszédet generálni. A fejlesztők, akik fel szeretnék használni a meglévő programjukhoz a rendszert egy API segítségével tudják megszólaltatni. Felhasználása részben ingyenes egy meghatározott bájtig, majd extra költséggel számol.

Amazon Polly

Az Amazon Polly egy felhőszolgáltatás, amely élethű beszéddé alakítja a szöveget, egyszerűen integrálható API műveletekkel. Több nyelven kínál számos élethű hangot, lehetővé téve multilingvális alkalmazások létrehozását, amelyek az ügyfelek számára legmegfelelőbb hangot használják. Emberszerű hangjai kifejező érzelmi és hanglejtési képességekkel rendelkeznek. Bár a hanggenerálás fizetős, a már létrehozott hangok ismételt lejátszása ingyenes. [16]

Pico2wave

A pico2wave a SVOX által készített TTS szolgáltatás, amely képes szöveget beszéddé alakítani. A szolgáltatás megfelelő működéséhez manuálisan kell telepíteni a pico2wave bináris fájlt, így nem kötött API-hoz. Használata ingyenes, azonban hanggenerálási minősége elmarad a versenytársaiétól. [17]

2.3 Nyelvmodell kiválasztásának folyamata

2.3.1 Célkitűzés és függőségek

A kutatás célkitűzése olyan Nagy Nyelvi Modellek kiválasztása, amelyek mély gépi tanulási módszerekkel képesek a kifejezéseket egy olyan térben pozícionálni, ahol a rendszer távolságmetrikák alapján értelmezheti az információkat. A cél, hogy a modell hatékonyan teljesítse a szövegértési és szöveggenerálási feladatokat, miközben magas színvonalú nyelvi megoldásokat nyújt, alkalmazkodva a felhasználói igényekhez, és elősegítve a produktivitást és az innovációt.

Adatgyűjtés és előkészítés

A projekt keretében vizsgált modellek már rendelkeznek előzetesen felhasznált adatokkal, azonban, ha az a szempont, hogy az alapértelmezettnél intelligensebben legyen képes összefoglalni egy forrás tartalmát, akkor az elvárás teljesítéséhez adatgyűjtést kell végezni és további tanítás elé kell vetni a modellt. Kiemelt szerepet fog itt kapni az adatok előkészítése.

Erőforrások biztosítása

Egy természetes nyelv feldolgozó modell tanításakor elhanyagolhatatlan az erőforrások biztosítása, ugyanis a modellek jelentős számítás kapacitást igényelnek ekkor. Általában igényük kifejezetten erős grafikus feldolgozó egységre és nagy mennyiségű memóriákra terjed ki.

2.3.2 A legnépszerűbb nagy nyelvi modellek

BERT

A Bidirectional Encoder Representations from Transformers (Bert) – Kétirányú Enkóder Reprezentációk Transzformerekből egy természetes nyelvi feldolgozási modell, amelyet a Google Research kutatói hoztak létre 2017-ben. [18]

A BERT nyelvmodell hatalmas mennyiségű tanítóadatot igényelt, ami kifejezetten a Wikipédia (~2,5 milliárd szó) és a Google Books (Google Könyvek) korpuszai (~800 millió szó) támogatásával valósított meg. Ezek a nagy információs adatkészletek hozzájárultak ahhoz, hogy a BERT nem csak az angol nyelvet, hanem a világunkat is mélyrehatóan megismerje. Azonban egy ekkora adatkészletre való feltanítás hosszú időt vesz igénybe. Szükségessé vált egy paradigma váltás, amivel gyorsíthatóak a folyamatok. A gyorsaságnövekedést az új transzformer architektúra hozta el, amely a Tensor Processing Units (TPU) – Tenzor Feldolgozó Egységeket használt fel. Ez az innovatív technológia a Google egyedi áramköre, amit kifejezetten a gépi tanulásra alkottak meg. A változtatás egy nagyon hatékony gépi tanulási párhuzamosítási stratégiává vált. Alapötlete, hogy a transzformerek figyelmi mechanizmust használnak a szavak közötti kapcsolatok megfigyelésére. Inspiráló forrása a népszerű 2017-ben megjelenő nagy

hatással rendelkező tanulmány, az „Attention Is All You Need” – „A figyelem minden, amire szükséged van”. Ezzel forradalmasította a Természetes Nyelv Feldolgozás terét, ugyanis megoldott az NLP leggyakoribb feladatai közül több, mint tizenegyet hatékonyabban, mint elődjei. A sikerét a transzformereken kívül az újszerű megvalósításai is előkészítették. [19] [20]

Továbbiakban létezik két innovációt jelentő technika a nagy nyelvi modellek világában:

- **Maszkolt Nyelvi Modell**: kétirányú tanulást végez a szövegből úgy, hogy egy mondatban letakart egy szót, és arra kényszerítette a BERT-et, hogy az elrejtett szó mindkét oldalán lévő szavakat használja fel a maszkolt szó előrejelzéséhez. Ezt korábban még senki sem próbálta meg! [19]
- **Következő Mondat Előrejelzése**: megjósolja, hogy egy adott mondat követi-e az előző mondatot vagy sem, ezzel nagyban segítve a mondatok közötti kapcsolatok megismerését. [19]

Llama

A Llama modellek a Facebook Meta vállalat által fejlesztett, nyílt forráskódú nagy nyelvi modellek, amelyek célja a mesterséges intelligencia elérhetővé tétele szélesebb kör számára. Működésük az előbbiekben ismertetett transzformer alapokon történik. [21]

A Llama modelleknek több verziója elérhető:

- **Llama 1:** Egy kisebb paraméterszámú modell, amely hatékonyabban működött, mint a korábbi nagyobb konkurens modellek. [22]
- **Llama 2:** A modellt 40%-kal több adatra tanították és kétszer akkora kontextushosszú, mint az elődje. Ez egy pontosabb és erőteljesebb nyelvi modellt jelent, amely emberszerű válaszokat tud adni. [21]
- **Llama 3.1:** Az előző verziókhoz képest szélesebb skálában elérhetőek a modellek, amelyek nagyobb rugalmasságot és testreszabhatóságot nyújtanak. [23]
- **Llama 3.2:** Az előző modellekhez képest multimodális képességeket kapott, amellyel képes lett egyszerre szöveges és vizuális adatok feldolgozására. Továbbá elérhetővé tette az optimalizált modelleket, amelyek mobil eszközökön is képesek futni. [23]

GPT

A Generative Pre-train Transformers (GPT) - Generetív Előtanított Transzformer, az amerikai OpenAI cég által kifejlesztett Nagy Nyelvi Modell, mely ugyan zárt forráskódú, azonban felhasználása engedélyezett egyéni programokban API kulcs segítségével havi díj ellenében. Működése szintén transzformer alakokon történik. [24]

A GPT modelleknek több verziója elérhető:

- **GPT-3:** A modell egy nagy paraméterszámmal rendelkező fejlett nyelvi modell, amely neurális hálózati gépi tanulásra épül. Az interneten található hatalmas mennyiségű szöveges adat feldolgozásával képes a bemeneti szöveget a lehető leghasznosabb eredmény előállítására optimalizálni.
  A modell képzéséhez különféle adatkészleteket, például a Common Crawl-t, a WebText2-t és a Wikipediát használták, gondosan megválasztott súlyozással annak érdekében, hogy hatékonyan támogassák a minták felismerését és a generatív előképzést. Ez a folyamat egy átgondolt tanítási stratégiát követ, amelyben a fejlesztők kérdésekkel tesztelik a modellt, majd a kapott válaszok alapján finomhangolják annak működését, biztosítva a lehető legjobb teljesítményt. [25]
- **GPT-3.5:** egyben tartalmazta a GPT-3 modellt és az OpenAI Cortex új fejlesztéseit a „text-davinci-002" és a „code-davinci-002" -t. Sokkal erőteljesebbnek tartották számon az előző verzióhoz képest. Továbbá motorként szolgált a népszerű ChatGPT számára.
- **GPT-4:** Az előző verzióhoz képest jóval szélesebb és sokrétűbb adathalmazon történt a tanítása, amely lehetővé tette a fejlettebb logikai gondolkodást és az összetett, több lépésből álló következtetések hatékonyabb végrehajtását. Ennek köszönhetően különösen jól teljesít kódolási, matematikai és döntéshozatali feladatokban. Emellett képes hosszabb párbeszédek vagy dokumentumok során is fenntartani a koherenciát, továbbá fejlettebb hibaellenőrzési mechanizmusokat alkalmaz az elfogultság minimalizálása érdekében.
  Ez a modell már képes fájlok, hanganyagok és weboldalak értelmezésére, lehetővé téve az interneten alapuló pontos tájékoztatást egy adott témában. [26]
- **GPT-4o:** Egy optimalizált verzió, amely kiemelkedő gyorsaságot kínál, miközben alacsonyabb számítási költséggel működik. Teljesítménye jelentős mértékben javult, és az eredmények magas színvonalát garantálja.

2.3.3 A feladathoz mérten választott modellek

A nagy nyelvi modellek széleskörű megismerése után kiválasztottam a modelleket, amelyeket a szakdolgozatom adatfeldolgozásához fogok használni.

Lokális adatfeldolgozás tekintetében a Llama 3.2 generációs modell bizonyult ideálisnak, mivel helyi gépen futtatható, elkerülve a távoli szerverek használatát, és kompakt mérete mellett is kiemelkedő teljesítményt nyújt.

A globális adatfeldolgozáshoz az OpenAI GPT-4o modellt választottam, mivel nagy paramétermérete révén kiemelkedő eredményeket nyújt, miközben távoli elérése minimális helyi erőforrásigényt jelent.

2.4 MI Ágensek ismertetése és szerepük a feladatban

A mesterséges intelligencia területén egy ágens egy olyan szoftveralkalmazási egység, amely képes felismerni és értelmezni a környezetét, valamint autonóm döntéseket hozni és lépéseket végrehajtani egy meghatározott cél vagy célcsoport elérése érdekében. Az ágens önállóan működik, tehát nincs közvetlen emberi irányítás alatt. [27]

2.4.1 Az ágensek jellemzőjük alapján

Az intelligens ágenseket számos jellemzőjük alapján tudjuk megkülönböztetni

Típusa alapján

A reaktív ágensek olyan entitások, melyek az azonnali környezeti ingerekre reagálva döntenek és hajtanak végre cselekvéseket. A proaktív ágensek kezdeményezően lépnek fel, és előre kigondolják lépéseiket a céljaik megvalósítása érdekében. [27]

Tevékenységeik környezete alapján

Az állandó környezetek fix szabályok szerint működnek, amelyek állandóak, míg a változékony környezetek folyamatosan változnak, így az ágenseknek alkalmazkodniuk kell az új helyzetekhez. [27]

Rendszerek ágensszám alapján

A többágenses rendszerek esetében számos ágens működik együttesen annak érdekében, hogy megosztott „erővel” teljesítsék a feléjük publikált igényt. A cél eléréséhez elengedhetetlen, hogy az egyedülálló ágens egységek képesek legyenek egymással történő kommunikációt lefolytatni és megosztani egyéni állapotukat. [27]

2.4.2 Az AI ágens felépítése

Az intelligens ágensek megértéséhez fontos ismerni az architektúrájukat. Egy ágens érzékelőkkel és cselekvést kiváltó operátorokkal van ellátva, például egy chatbot hangérzékelő rendszerrel és feldolgozó algoritmussal működik. Az ágensprogram biztosítja a működést, míg az ágensfüggvény az érzékelési szekvenciát cselekvéssé alakítja. Az ágens működése így az architektúra és az ágensprogram összefüggéséből ered. [27]

2.4.3 Az ágensek típusai

Egyszerű reflex ágens

A legyorsabb döntést hozó ágensek, melyek gyorsasága abban rejlik, hogy csak az aktuális észlelési állapot alapján reagálnak. Működésük esemény-felvétel-művelet szabályrendszeren alapul, ahol a felhasználó kezdeményezi az eseményt, majd az ágens a korábban ismertetett módon reagál. [28] [29]

Modell alapú reflex ágens

Előnye az előzmények figyelembevételében rejlik. Képes olyan közegben dolgozni, mely számára kezdetben ismeretlen. Működésük feltétel-művelet alapú, mely lehetővé teszi, hogy átfogóan döntsenek a cselekvésről, kiegészítve a belső állapotukkal, a jelenlegi környezeti állapotukkal és észlelési történetükkel. [30]

Cél alapú ágens

Ezek az ágensek a mesterséges intelligencia alkalmazásával célorientált döntéseket hoznak, kibővítve a modellalapú és reflex ágensek lehetőségeit. Proaktívan részt vesznek a keresés és tervezés folyamatában, ahol különböző forgatókönyveket és cselekvési lehetőségeket mérlegelnek, hogy a legmegfelelőbb megoldást érjék el. [31]

Hasznosság alapú ágens

A hasznosság alapú ágensek a célalapú ágenseknél fejlettebb funkcionalitást kínálnak, mivel hasznossági mérésekkel értékelik a sikert. Nemcsak a célokat veszik figyelembe, hanem optimalizálják a cél elérésének legideálisabb módját is. Különösen előnyösek, ha egy cél többféleképpen érhető el, vagy ha számos tényező befolyásolja a folyamatot, fejlett döntéshozatali képességeik révén biztosítva az optimális eredményt. [31]

Tanuló ágensek

Az AI-alapú tanulóágensek tanulási algoritmusok segítségével fokozatosan javulnak, miközben egyre jobban megértik a környezetüket. A tanulási elem a teljesítmény-visszajelzések alapján optimalizálja az ágens működését, négy fő összetevőre támaszkodva:

- **Tanulási elem:** a környezeti tapasztalatok során tanul és fejlődik.
- **Kritikai elem:** teljesítményértékelést készít az ágensről szabványok alapján.
- **Végrehajtó elem:** az előzők birtokában döntést hoz a végrehajtásról.
- **Problémageneráló elem:** új tapasztalatokat kezdeményez a fejlődés érdekében.

Ez a struktúra biztosítja az ágens folyamatos fejlődését és alkalmazkodóképességét. [27] [29] [32]

2.4.4 Ágensek szerepe szakdolgozatban

A mesterséges intelligencia ágensek kiemelt szerepet kapnak a szakdolgozatomban, mivel hatékonyan képesek kinyerni és értelmezni a dokumentumok esszenciális információit, valamint megbízható összefoglalókat készíteni. Ezek az eszközök biztosítják, hogy a tartalom a felhasználó által preferált nyelven érthető legyen, miközben rövidített kivonatokkal segítik az eredeti szöveg átlátását. Az ágensek alkalmazása egyben lehetőséget teremt a dokumentum részletesebb megismerésére felmerülő kérdések megválaszolásával, így a legkifinomultabb megoldást kínálják a feladat komplexitására.

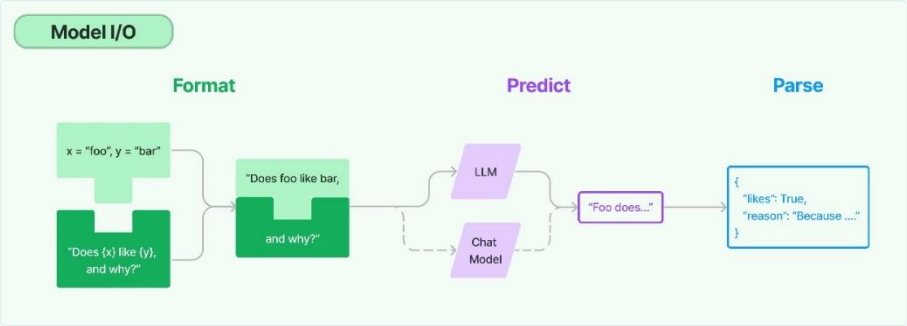
2.5 Langchain keretrendszer bemutatása

A Langchain egy nyílt forráskódú moduláris keretrendszer, amely megkönnyíti az mesterséges intelligencia alapú nyelvi alkalmazások fejlesztését, beleértve a gépi tanulást is. Jelenleg Pythonban és JavaScriptben érhető el. Ahogyan a neve is tartalmazza Lang (a language rövidítése, jelentése: [beszélt] nyelv), Chain(jelentése: lánc), tökéletesen mutatja, hogy a működésének alap pillére az egységek egymáshoz való láncolásában rejlik. A segítségével írt programokat globális nagyvállalatok, induló vállalkozások és magánszemélyek is használják, így sokoldalú eszköz a számítástechnika területén. [33]

Arra tervezték, hogy egyszerűsítse az alkalmazások létrehozását a nagy nyelvi modellek (LLM) és különböző komponensek zökkenőmentes összecsatolt felhasználásával, valamint az olyan erőforrások beépítésével, mint az API-k és adatbázisok. A keretrendszer szabványos interfészt biztosít a „láncokhoz”, ezzel együtt számos integrációt kínál, ami lehetővé teszi a fejlesztők számára, hogy könnyedén összekapcsolják a Langchain -t más rendszerekkel és technológiákkal. Előnyei közé sorolható, hogy képes kezelni egy alkalmazás teljes életciklusát, az adatgyűjtéstől és feldolgozástól kezdve egészen az eredmény bemutatásáig. Ezt a szaknyelv end-to-end (végtől-végig) megoldásként tartja számon. [34] [35] [36]

2.5.1 A Langchain komponensei

Modell I/0



2.3. ábra: Langchain modell [37]

A Langchain Modell I/O, a nyelvi modellekkel való interfész létrehozásának alapvető eleme. Számos építőelemmel rendelkezik.

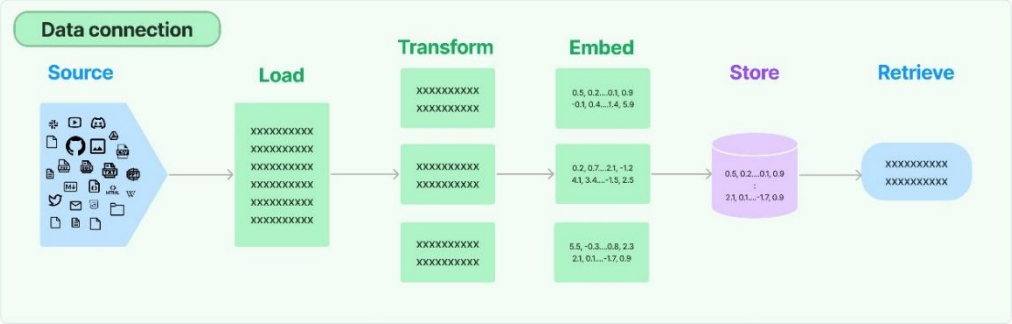
Ezen belül a "Prompts" (utasítások) kulcsszerepet játszanak, mint bemenetek, melyeket a felhasználó ad a modellnek. Ezek segítik a modellt a kontextus megértésében és releváns nyelvi kimenet generálásában. A bemenetek lehetnek például kérdések, mondatok befejezése, melynek célja a hatékony reakció és értelmes tartalom létrehozás elősegítése. [37] [38]

A rendszer másik fontos építőeleme az LLM (Nagy Nyelvi Modell) alapú modellek, amelyeknek bemeneti igényük szöveges karakterláncra terjed ki, majd szöveges

karakterláncot generálnak vissza. Hasonlóan működnek a Chat modellek is, azonban bemenő adatként egy csevegőüzeneteket tartalmazó listát fogadnak el, és egy csevegési üzenettel térnek vissza. [37]

Az Output Parsers (kimeneti elemzők) a Langchainben olyan osztályok, amelyek segítenek strukturálni a nyelvi modellek válaszait. Különféle módszerek állnak rendelkezésre, amelyek megkönnyítik ezen folyamatokat. Lehetőségünk van lekérni a formázási utasításokat, ami információt tartalmaz a nyelvi modell kimenetével kapcsolatos formázási lehetőségről. Ezenkívül kérhetjük a rendszert, hogy szervezze struktúrába a nyelvi modell visszaadott karakterláncát és végezzen elemzést belőle. [39]

Retrieval (visszakeresés)



2.4. ábra: Visszakeresési folyamat bemutatása [41]

A visszakeresés a Langchain technológiai platform egyik kulcsfontosságú összetevője. A Langchain Retrieval szemantikus keresésen alapul, ahol minden dokumentumhoz egy numerikus vektort (beágyazást) rendelnek, majd ezeket a vektorokat egy speciálisan optimalizált adatbázisban tárolják. A rendszer célja, hogy hatékonyan és gyorsan képes legyen lekérdezéseket végezni ezen vektorok alapján, lehetővé téve a felhasználók számára a releváns információk gyors és pontos visszakeresését. [40]

A Retrieval-augmented Generation (RAG) egy mesterséges intelligencia keretrendszer, melynek célja a nagy nyelvi modellek által generált válaszok minőségének javítása. Ennek érdekében a rendszer külső tudásforrásokat használ fel, hogy kiegészítse az LLM (Nagy Nyelvi Modell) belső információ-reprezentációját. A Document loaders (dokumentum betöltők) segítségével a Langchain képes számos különböző forrásból betölteni dokumentumokat, és több mint száz különböző integrációval rendelkezik nagy szolgáltatókkal, lehetővé téve gyakorlatilag az összes dokumentum típus betöltését (pl. HTML, PDF stb.). Ezáltal a rendszer gazdagabb tudásbázissal rendelkezik, amit felhasznál a generált válaszok optimalizálásához és pontosabbá tételéhez. [41]

A visszakereső komponens számos olyan építőelemet tartalmaz, amelyek együttesen hozzájárulnak a hatékony információkereséshez és feldolgozáshoz. A Document

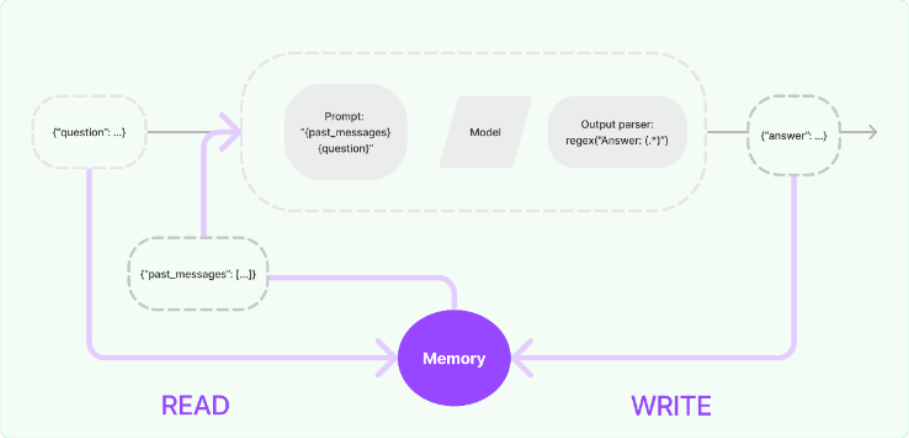
transformers (dokumentum átalakítók) fontos szerepet játszanak a releváns dokumentumrészek lekérdezésében, különféle transzformációs lépések segítségével. Ez magában foglalja a nagy dokumentumok kisebb darabokra történő felosztását is. A Text embedding models (szövegbeágyazási modellek) felelősek a dokumentumok beágyazásának létrehozásáért, amelyek rögzítik a szöveg szemantikai jelentését, lehetővé téve a hasonló szövegrészek gyors és hatékony azonosítását. Ahogy növekszik a beágyazások száma, felmerül az igény a hatékony tárolásra és keresésre alkalmas Vector stores (vektor tárolók) adatbázisokra. Ezek lehetővé teszik az beágyazások hatékony tárolását és keresését. Az adatok vektor adatbázisban való elhelyezése után, a Retrievers (visszakeresők) felelősek a releváns információk lekérdezéséért. A Langchain számos lekérdezési algoritmust támogat, ezen a területen kiváló értéket nyújtva. Ez a rendszer egységesen kombinálja ezeket a komponenseket, hogy támogassa a szemantikus keresést és a hatékony információfeldolgozást. [41]

A visszakeresők, mint a Parent Document Retriever és a Self Query Retriever, kardinális jelentőségű elemek a hatékony információkeresési folyamatokban. A Parent Document Retriever lehetővé teszi a több beágyazás létrehozását szülődokumentumonként, kifinomultan kezelve a kisebb darabokat, és így biztosítva a precíz keresést, miközben szélesebb kontextust nyújt vissza. A Self Query Retriever kifejezetten a felhasználói kérdések komplexitására fókuszál, figyelembe véve az olyan utalásokat, amelyek nem csak szemantikaiak, hanem logikai kapcsolatokat is kifejeznek. Az önlekérdezés lehetővé teszi a lekérdezés szemantikai részének elemzését a lekérdezésben lévő többi metaadatszűrőből, hozzájárulva a rendszernek a pontos és releváns válaszok generálásához a felhasználói kérdésekre. Ezek az összetevők egységesen működnek együtt a Langchain platformon, hogy támogassák a szemantikus keresést és a hatékony információfeldolgozást. [41]

Chains („láncok” interfészek)

A Langchain modulok közül a lánc interfész két fő keretrendszert kínál komponensek “összekapcsolására”. Az egyik a régebbi egyszerű lánc interfész, míg a frissebb a Langchain Expression Language (LCEL) - Langchain kifejező nyelve. AZ LCEL leglátványosabb része, hogy intuitív és olvasható szintaxist biztosít az összetételhez. Emellett első osztályú támogatást nyújt többek között a streaminghez, aszinkron hívásokhoz, kötegeléshez, párhuzamosításhoz és nyomkövetéshez. [42]

Memória



2.5. ábra: Memória: Írás és olvasás folyamata [43]

A legtöbb LLM-alkalmazás párbeszédes felülettel rendelkezik. A beszélgetés lényeges eleme, hogy képes legyen hivatkozni a beszélgetés során korábban bemutatott információkra. A párbeszédes rendszernek képesnek kell lennie arra, hogy közvetlenül hozzáférjen a múltbeli üzenetek néhány „emlékéhez”. Egy összetettebb rendszernek rendelkeznie kell egy olyan kiterjesztett modellel, amelyet folyamatosan frissít lehetővé téve számára például az entitásokról és kapcsolatairól szóló információk karbantartását. Ezt a képességet a múltbeli interakciókra vonatkozó információk tárolására „memóriának” nevezzük. A LangChain számos segédprogramot biztosít a rendszer memória hozzáadásához. Ezek a segédprogramok önmagukban is használhatók, vagy zökkenőmentesen beépíthetők egy „láncba”. [43]

A memória két alapvető funkciót lát el: az írást és az olvasást. A rendszer egy adott futás során ugyanennyiszer folytat interakciót a memóriával. Az olvasás az alap felhasználói bemenetek után és az alaplogika végrehajtása előtt történik. Az írás az alaplogika után, de a válasz visszaadása előtt következik be. [43]

Agent (ágens)

Az ágensek alkalmazása forradalmian új megközelítést jelent a műveletsorozatok tudatos kezelésében, elősegítve a rugalmas és intelligens válaszokat a felhasználói bevitelre. Az ágensek érvelésének eredményessége a nyelvi modellek használatának köszönhető, mely központi motorként működik. Sokszínűségét az jelenti, hogy az ágensek eltérő gondolkodási stílusúak és különböző módon értelmezik a bemeneteket és kimeneteket. Egyszerű használatát a Langchain teszi lehetővé, megkönnyítve az eszközök egyéni céloknak megfelelő alakítását és a hozzáférést. [44]

Callbacks (visszahívás)

A Langchain moduljai között megtalálható a Callbacks rendszer, mely lehetőséget biztosít a visszakapcsolódásra az LLM alkalmazás különböző szakaszaiban. Ez hasznos funkció lehet naplózás, monitorozás, adatfolyamok és más feladatok számára. Ezekre az eseményekre az API-n keresztül elérhető callbacks argumentum segítségével lehet feliratkozni. [44]

2.6 Docker konténerizáció

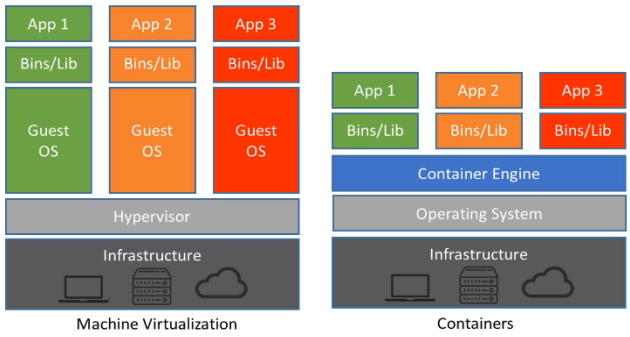
A Docker egy nyílt platform alkalmazások fejlesztésére, szállítására és futtatására. Segítségével elválaszthatóak az alkalmazások az infrastruktúrától, így könnyedén és gyorsan lehet szoftvereket szállítani. A Docker, Go programozási nyelven íródott és kihasználja a Linux kernel számos funkcióját a funkcionalitás biztosításához. [45]

2.6.1 Docker konténerizáció és virtuális gép összehasonlítása

A Docker és a Virtual machine (virtuális gép) működése korrelál, ugyanis közös alapjukat a virtualizáció képzi, azonban számos különbség is felfedezhető a két technológia között, mint például az operációs rendszer biztosítása. E tekintetben számos oldalról lehet vizsgálni az előnyöket és hátrányokat. A Docker „könnyebb súlyú”, vagyis tárhelyigény szempontjából kevésbé kimerítő a jelenléte. Ez annak köszönhető, hogy a gazdagépen lévő operációs rendszer kerneleit használja minden container (konténer) egy elkülönített környezetben, míg a virtuális gép minden virtuális példánynak létrehoz egy saját operációs rendszert ezzel növelve a kapacitási szükségletet. A méret egyenes arányossággal határos a teljesítménnyel, így a rendszer indításának bajnokaként a Dockert tartják számon, ugyanakkor számos hátránynak nevezhető dolog származik a gazdagépen lévő kernelek használatából. Ezek közé sorolható a biztonság és kompatibilitási oldal. A Docker legjobban Linux disztribúciókkal működik, míg a versenytársa saját operációs rendszer telepítése miatt közömbös eme kérdésben.

2.6.2 Docker architektúra

A Docker, kliens-szerver architektúrát használ. Az architektúra három fő komponensből



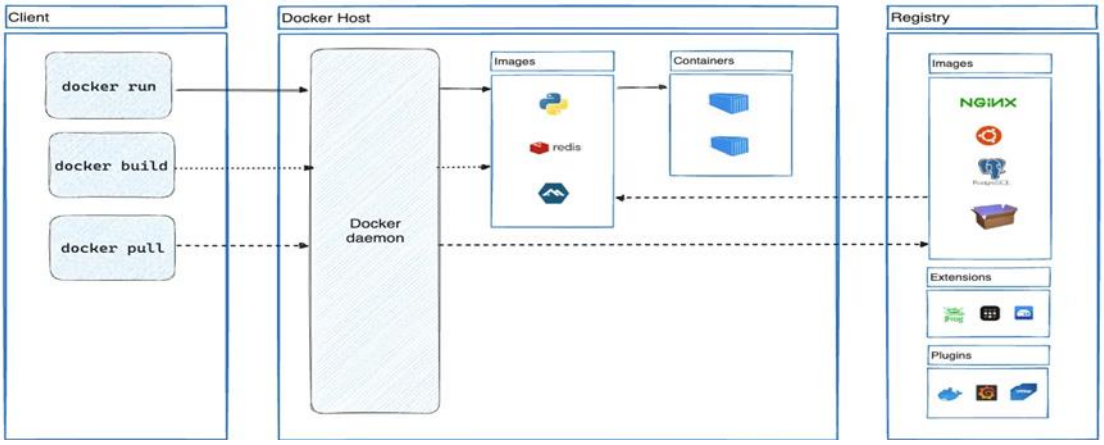
2.6. ábra: Virtuális gép és konténerizált környezet összehasonlítása [63]

áll, Client (kliens), Docker Host (szerver) és Registry (Docker képnyilvántartó).

A Docker-kliens lehetővé teszi a felhasználók számára, hogy kapcsolatba lépjenek a Dockerrel. A Docker Host teljes környezetet biztosít az alkalmazások végrehajtásához és futtatásához, valamint tartalmazza a Docker Daemon-t, képfájlokat, konténereket, hálózatokat és konténer kötetetket. Végül a Registry, mely olyan szolgáltatások összessége, amik olyan helyeket biztosítanak, ahonnan a Docker image-ket (Docker képfájlokat) és bővítményeket tárolni lehet és letölteni. [46]

Továbbiakban részletesen kifejtem az egyes komponenseket és a bennük található Docker eszközöket.

2.6.3 Docker objektumok



2.7. ábra: Docker architektúra bemutatása [45]

Docker Client (kliens)

Az elsődleges módja annak, hogy a Docker-kliens kapcsolatba lépjen a Dockerrel. A parancsokat a Docker Daemon kezeli, melyből akár több is várhatja a kliens kéréseit. Kommunikációjuk REST API-n keresztül, UNIX socketeken vagy hálózati interfészen keresztül történik. [45]

Docker Daemon

A Docker egyik beépített eszköze, feladata várni a Docker API kéréseket és kezelni a Docker objektumokat, mint például image (kép fájlok), containers (konténerek), hálózatok és volumes (kötetek, a Docker konténerek által használt adattároló egységek). Ezen felül a Daemon példányok, más Daemon beépített eszközökkel is képesek kommunikálni. [45]

A konfigurálásához előnyben részesített lehetőség a JSON konfigurációs file használata, ugyanis így az összes futtatásához szükséges adatot egy helyen lehetséges segítségével tárolni. Az indítás folyamatának megkezdésekor elengedhetetlen a flag (jelölő) használata, mely a "dockerd" kulcsszó beírásával érhető el. A Docker Daemon az összes adatát egy úgynevezett Daemon adatkönyvtárban tárolja, mely nyomon követi az összes előbbiekben felsorolt Docker objektumot. [47]

Docker file

A Docker automatikusan képes images (képfájlokat) létrehozni a helyes Dockerfile beolvasását követően. A Dockerfile egy szöveges dokumentum, amely tartalmazza az

összes parancsot, amelyet a felhasználó meghívna a parancssorban egy képfájl összeállításához. [48]

Docker Container (konténer)

A konténer a Docker image futtatható példánya. [45]

A konténer egy szabványos szoftveregység, amely összecsomagolja a kódot és annak összes függőségét, így az alkalmazás gyorsan és megbízhatóan fut az egyik számítógépes környezeten vagy bármely másikon. A Docker konténer egy könnyű, önálló, végrehajtható szoftvercsomag, amely mindent tartalmaz, ami egy alkalmazás futtatásához szükséges: kód, futási környezet, rendszereszközök, rendszerkönyvtárak és beállítások. [[49]

A Docker API vagy CLI segítségével létrehozhatja, elindíthatja, leállíthatja, áthelyezheti vagy törölheti a konténert. Egy konténert csatlakoztathat egy vagy több hálózathoz, tárhelyet csatolhat hozzá, vagy akár új képet is létrehozhat az aktuális állapota alapján. Alapértelmezés szerint egy konténer viszonylag jól el van szigetelve a többi konténertől és a gazdagéptől, azonban ez későbbiekben manuálisan is finomítható. A konténerek csak a képfájlban meghatározott erőforrásokhoz férhetnek hozzá, kivéve, ha további hozzáférést adnak meg a képfájl konténerbe építésekor. [45]

A konténer csomag meghatározó jellegű, hiszen mikor eltávolításra vagy „kilövésre” kerül, akkor minden adat elveszik belőle. [46]

Docker Image (képfájl)

Az image egy írásvédett sablon a Docker konténer létrehozására vonatkozó utasításokkal. A Docker közösség által már számos publikált megoldás elérhető bárki számára, azonban van lehetőség saját image létrehozásra is. A saját image kialakításához létre kell hozni egy Docker-fájlt egy egyszerű szintaxissal, amely meghatározza a kép elkészítéséhez és futtatásához szükséges lépéseket. Felépítésük rétegesen történik, ugyanis gyakran előfordul, hogy egy image egy másik image-n alapul némi testre szabással. Amikor módosítja a Docker-fájlt és újraépíti az image-t, csak a megváltozott rétegek épülnek újra. Ez része annak, ami miatt a képfájlok olyan könnyűek, kicsik és gyorsak, összehasonlítva más virtualizációs technológiákkal. [45]

Docker Registry (Docker képnyilvántartó)

A konténerizált környezet egyik legmeghatározóbb előnye, hogy egyszerűen szállíthatóak a szoftverhez szükséges adatok a számítástechnikai eszközök között. A Docker Registry a Docker által biztosított, önállóan kezelt megoldás, kifejezetten a könnyű image elérhetőség céljára kifejlesztve. A konténer registry tárolja az egyes konténer image-ket a felépítésük után, valamint a kapcsolódó metaadatokat (például a verziókat, címkéket, létrehozási dátumokat, szerzőket és még a biztonsági vizsgálat eredményeit is). A Docker

a felhőben futó felügyelt és biztonságos image-k eléréséhez a Docker Hubot kínálja, mely jelenleg a legelterjedtebb rendszerleíró adatbázis. [50]

A programozó szabadságát fokozza, hogy bár a Docker alapértelmezetten a Docker Hub megoldásai között keresi az image-t, ugyanakkor nem határozza meg követelményként alkalmazását, így saját registry-t is lehet definiálni.

Docker Compose

A Docker Compose egy olyan eszköz, mely segíti a több konténerből álló alkalmazások meghatározását és megosztását. A Compose használatának jelentős előnye, hogy lehetővé teszi az alkalmazások összetevőinek definícióját egyetlen (YAML) fájlban, amely a projekt gyökérkönyvtárában található. Ez a fájl verziókezelhető, ami segít a változások nyomon követésében. Továbbá, a Docker Compose használata elősegíti a közösségi hozzájárulásokat a projekthez, mivel más fejlesztők könnyen megérthetik és módosíthatják az alkalmazás struktúráját. Az adattárat csak klónozni kell, majd a Compose segítségével indítani az alkalmazást. Gyakran találkozhatunk olyan GitHub/GitLab projektekkel, amelyek éppen ezt a megközelítést alkalmazzák. [51]

Docker Engine (Docker „motor”)

A Docker Engine a Docker alkalmazások felépítésének és konténerizációjának kulcsfontosságú „motorja”. Ez az alapvető cselekvést indikáló egység rendelkezik olyan API-kkal, amelyek lehetővé teszik a programok számára a kommunikációt a Docker-Daemonnal és utasításainak kiadását. A felhasználók részére parancssori felületet (CLI) biztosít, amelyeken keresztül könnyedén végezhetik el a konténerizációs feladatokat. Ezzel az Engine lényeges szerepet játszik a Docker ökoszisztémában, lehetővé téve a hatékony és egyszerű alkalmazásfejlesztést és üzemeltetést. [52]

Docker Network (Docker hálózat)

Fő célja kapcsolatot kiépíteni a konténerek között ezzel létrehozva az egymással való kommunikációjukat. Amennyiben két konténer ugyanazon a számítógépen fut, egymás közötti kommunikációra képesek anélkül, hogy a számítógép portjait ki kellene publikálni számukra. A Docker segítségével könnyedén kezelhető a Docker-host bármilyen platformon, függetlenül attól, hogy Windows vagy Linux operációs rendszert futtat a környezet. [53]

Docker Volume (Docker kötetek)

A Docker volume a Docker konténerek által generált és használt adatok megőrzésére szolgáló mechanizmus. A Docker kötetek előnyei között szerepel, hogy könnyedén készíthető róluk biztonsági mentés és egyszerűen hordózhatóak. [54]

Docker Swarm

A Docker Swarm egy nyílt forráskódú konténerorkestrációs eszköz, amely lehetővé teszi a konténerek egységes kezelését és skálázását egy vagy több gépen. A Docker Swarm lehetővé teszi, hogy a konténereket egyetlen logikai egységként kezelje, amelyet "service" néven ismerünk. A Docker alapértelmezetten magában foglalja a Swarm eszközt, így előnyt kovácsol számos kompatibilitási szempontból. Működése zökkenőmentes a meglévő Docker eszközökkel és támogatja ezek csoportjainak összehangolását, biztosítva az automatizált terhelés elosztást. Az előnyökkel egyidejűleg érdemes figyelembe venni, hogy bizonyos esetekben ezek a pozitív aspektusok potenciális nehézségeket is magukkal hozhatnak, ugyanis erősen kötődnek a Docker API-hoz, ami a testre szabási lehetőségeiket szűkíti. [55]

3 A RENDSZERTERV BEMUTATÁSA

A feladat megvalósításához szükséges elemeket moduláris szerkezetben mutatom be, amely elősegíti a könnyebb áttekinthetőséget és értelmezhetőséget. Ez a megközelítés biztosítja, hogy az egyes komponensek önállóan is érthetőek legyenek, miközben egyértelműen illeszkednek a rendszer egészébe.

3.1 Adatbeolvasó modul

A rendszer célja

A rendszer célja egy adott könyvtárban található fájlok szöveges tartalmának kinyerése, különböző a leggyakoribb fájl típusok támogatásával. A kinyert szövegek egy strukturált adatszerkezetben kerülnek tárolásra további feldolgozás vagy felhasználás céljából.

Architektúra

- Bemeneti adat: egy karakter sorozat, mely az ideiglenes könyvtár elérési útvonalát mutatja.
- Kimeneti adat: egy szótár, mely tartalmazza a beolvasott fájlok nevét és tartalmát kulcs értékpár formájában.

Technológiai háttér

- Programozási nyelv: Python.
- Szövegfájlok feldolgozása: PyPDF2, python-docx, ebooklib, BeautifulSoup.
- Audio- és videófájlok feldolgozása: pydub, moviepy, SpeechRecognition.
- Optikai karakterfelismerés: pytesseract, PIL.
- Adatstruktúrák: JSON.
- Fájlműveletek és tömörített fájlok kezelése: zipfile, os, shutil.

Fő funkciók

- Fájlok eltárolása egy ideiglenes mappában a rendszer gyökérmappájában, majd egyenként innen beolvasni és átalakítani a kívánt formátumra. Majd az összes fájl beolvasása után a mappa törlése tartalmával együtt.
- Támogatni a leggyakoribb fájl típusokat:
    - Szövegfájlok: (PDF, DOCX, TXT, JSON, HTML, EPUB).
    - Hangfájlok: (MP3, WAV).
    - Videófájlok: (MP4, WMV).
    - Képfájlok: (PNG, JPG, JPEG).
    - ZIP archívumok (rekurzív feldolgozás a kicsomagolt fájlokból).
- Nem támogatott formátumok azonosítása hibakezelés szempontjából.

3.2 Adatgyűjtési modul

A rendszer célja

A rendszer célja, hogy szöveges dokumentumokat dolgozzon fel, oly módon, hogy képes legyen átvetíteni az adatokat egy vekoriális térbe. Továbbá szükséges olyan funkciókkal rendelkeznie, ami egy RAG rendszer adatbázis műveleteit képes kielégíteni.

Architektúra

- Bemeneti adat: az előző modulban elkészített szótár.
- Kimeneti adat: műveletek sikerességének visszajelzése.

Technológiai háttér

- Programozási nyelv: Python.
- Adatbázis-kezelés: Qdrant Python SDK.
- Embeddingek generálása: OpenAI (globális) vagy „all-MiniLM-L6-v2” (lokális).
- Fájlműveletek és azonosítók: os, uuid.
- Logging és hibakezelés: Python logging modul.

Fő funkciók

- Vektoradatbázis létrehozása és törlési műveletei.
- Szöveg előfeldolgozása (darabolás, vektor hozzárendelés).
- Adatok feltöltése az adatbázis megfelelő tárolórendszerébe.
- Hasonlósági keresés.
- Reakciók eltárolása a cselekvő megnevezésével.

3.3 Nagy nyelvi modellt biztosító modul

A rendszer célja

A rendszer célja nagy méretű szöveges dokumentumok automatikus összefoglalása és releváns válaszok generálása a felhasználói kérdésekre.

Architektúra

- Bemeneti adat: Az első modulban elkészített szótár; karaktersorozat a fogalmazás módjáról; karaktersorozat a vektoradatbázis gyűjteményének nevéről.
- Kimeneti adat: egy karaktersorozat, mely kifejezetten az adott feladatra generált szöveget tartalmazza. (fogalmazás vagy válasz a kérdésre).

Technológiai háttér

- Programozási nyelv: Python.
- Nyelvi modell: OpenAI GPT-4o, LLAMA (meta-llama/Llama-3.2-1B).
- Adatbázis-kezelés: Qdrant vektoradatbázis.

Használt könyvtárak

  - OpenAI SDK (GPT-4o API hívásokhoz).
  - asyncio (aszinkron feldolgozás támogatása).
  - tiktoken (tokenek kezeléséhez).
  - dotenv (környezeti változók betöltésére).
  - torch (neurális hálózatok számításai).
  - re (szövegfeldolgozás regex-szel).
  - logging (rendszeresemények naplózására).
  - Egyedi modulok: Szövegfordító ágens, Utasítás készítés, Vektoradatbázis interfész.

Fő funkciók

- Dokumentumok komplex összefoglalása.
- Felhasználói kérdések megválaszolása.
- Kommunikáció a vektoradatbázissal.

3.4 Láncolat modul

A láncolatok célja

A szoftverfejlesztés során a láncolatok alkalmazásával, amelyek kisebb egységekből, úgynevezett mikroszolgáltatásokból épülnek fel, lehetőség nyílik összetettebb programok létrehozására. Ezek a láncolatok nemcsak a rendszer átláthatóságát és hibakezelését segítik elő, hanem hozzájárulnak egy biztonságosabb és újrahasznosítható megoldás kialakításához is. Továbbá, ezen struktúra lehetővé teszi a program komplexitásának fokozatos és egyszerű bővítését, miközben fenntartja a hatékonyságot és a stabilitást.

Architektúra

Bemeneti adat: szótár az adatról vagy láncról (elnevezés, tartalom).

Kimeneti adat: szótár a láncszem eredményről (elnevezés, eredmény).

Technológiai háttér

Láncolatokat elősegítő könyvtár használata: Langchain

Használt láncok

- Alap láncolat: meghívja a definiált mikroszolgáltatásokat és futtatja. (Láncszemek).
- Szekvenciális láncolat: egymás után fűzi a láncszemeket és összetett láncolatot alkot belőlük.

3.5 Felhasználói felület modul

A rendszer célja

Az elkülönített felhasználói felület elsődleges célja, hogy a felhasználók számára lehetővé tegye a szükséges feladatok elvégzését anélkül, hogy programozói ismeretekkel kellene rendelkezniük. A rendszer egy intuitív grafikus felületet biztosít, amelyen keresztül a felhasználók hatékonyan végezhetik munkájukat, miközben folyamatos és pontos visszajelzéseket kapnak az egyes folyamatokról, ezzel elősegítve a tájékozott döntéshozatalt és a gördülékeny működést.

Architektúra

Bemeneti adat: Az adatok széles skáláját várja, mind a felhasználótól, mind a kiszolgálói oldaltól.

Kimeneti adat: Általában egy reakció, amit a számítógép képernyőjén megjelenít.

Technológiai háttér

- A webfelület elkészítéséhez az irodalomkutatás tekintetében Sveltekit keretrendszert választottam ki, amely modern és teljesítményorientált.
    - A keretrendszer a következő nyelveket támogatja.
        - Javascript (cselekvések előidézése)
        - CSS (stílus leíró nyelv)
        - HTML (jelölőnyelv)

Fő funkciók

- Grafikus felület biztosítása a felhasználónak.
- Adatok továbbítása a kiszolgálói oldalnak.

3.6 Kiszolgálói oldal modul

A rendszer célja

Kapcsolatot biztosítani a felhasználói felület és a program logikai rendszere között.

Architektúra

Bemeneti adatok: kérések a felhasználó felület oldaláról.

Kimeneti adatok: a kérésre adott választ tartalmazzák, amely lehet a sikertelen műveletről szóló értesítés, vagy a sikeresen végrehajtott folyamat eredményének visszaküldése.

Technológiai háttér

A két oldal kapcsolatára az irodalomkutatás tekintetében FastAPI keretrendszert választottam ki, amely modern és nagy teljesítményű aszinkron hívásokat tesz lehetővé.

- Használt könyvtár
    - fastapi (adat átvétel és küldés funkcióinak kielégítése érdekében).

Fő funkciók

- Végpontok elkészítése.
- Kapcsolattartás a Frontend és Backend között.

3.7 Konténerizációs modul

A rendszer célja

A rendszer célja egy mesterséges intelligenciára épülő alkalmazás konténerizációja, amely különböző micro-service komponenseket használ.

Előnyök

- Egyszerű telepíthetőség: Az alkalmazás gyorsan és egyszerűen futtatható legyen bármilyen környezetben.
- Skálázhatóság: Az egyes micro-service komponensek külön-külön skálázhatók legyenek.
- Modularitás: A különálló komponensek egymástól függetlenül fejleszthetők és frissíthetők.
- Konzisztencia: Az alkalmazás mindig ugyanúgy működjön minden környezetben (lokális, teszt, éles).

Technológiai háttér

- Docker kliens (parancssori eszköz)
- Docker fájl (utasításokat tartalmazó szöveges fájl)
- Docker Compose (deklaratív konfigurációs fájl)

4 MEGVALÓSÍTÁS

4.1 Fejlesztőkörnyezet bemutatása

A szakdolgozatom gyakorlati részét képező Tartalom-összefoglaló Mesterséges Intelligencia program fejlesztését Visual Studio Code környezetben valósítottam meg, míg a program futtatását Linux alrendszer alatt végeztem.

Visual Studio Code

A Visual Studio Code, a Microsoft által fejlesztett integrált fejlesztőkörnyezet, hatékony megoldást nyújt összetett programok létrehozására. Széleskörű nyelvtámogatása – például Python, Svelte, TypeScript és Docker – kiválóan illeszkedett a szakdolgozatomban használt technológiákhoz. Beépített terminálja, Git-kezelése és bővítményei jelentősen támogatták a fejlesztési folyamatot. Átláthatósága, rugalmassága és professzionális eszköztára ideális választássá tette a tartalom-összefoglaló program megvalósításához. Telepítése egyszerűen megvalósítható a Microsoft hivatalos weboldaláról. [56]

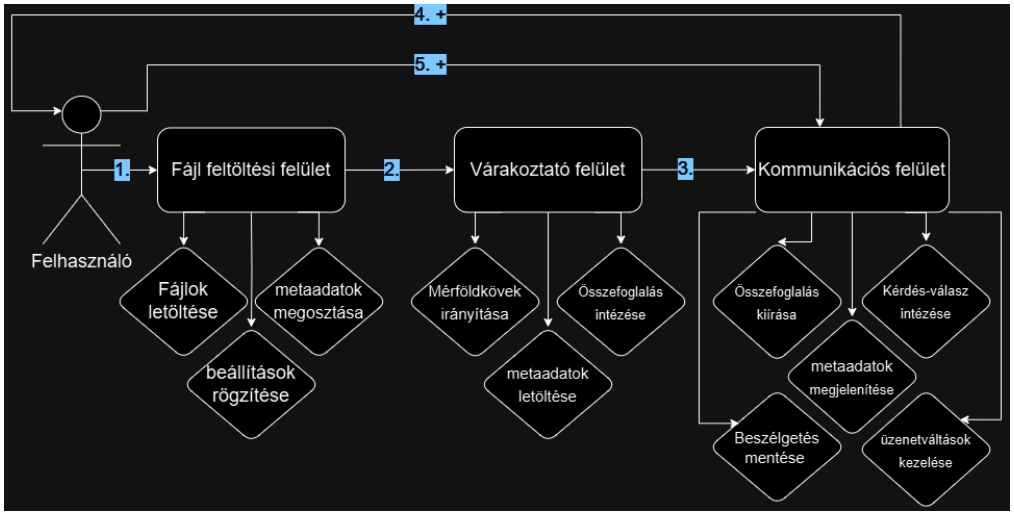
Docker Desktop – Docker asztali alkalmazás

A Docker Desktop egy intuitív grafikus felületet kínál, amely megkönnyíti a konténerek létrehozását, indítását, leállítását és kezelését, minimális parancssori ismeretekkel is. Az alkalmazás magában foglalja a Docker CLI-t, a Docker Engine-t, valamint az összes szükséges eszközt, telepítése pedig egyszerűen elvégezhető a hivatalos Docker weboldalon elérhető telepítőfájl segítségével. [57]

Windows Subsystem for Linux 2 (WSL2) – Windows alrendszer Linuxhoz 2

A WSL2 lehetővé teszi a Linux környezet natív futtatását Windows operációs rendszeren, biztosítva a Linux kernel teljesítményét. Ez előnyös a fejlesztési folyamatokban, különösen a Docker-alapú virtualizáció és a Python-fejlesztés eszközeinek támogatásában. Gyors és stabil működése révén ideális fejlesztői környezetet nyújt, hozzájárulva a szakdolgozatom alkalmazásának sikeréhez. Telepítése egyszerűen megoldható a Docker asztali alkalmazás megfelelő beállításával. [58]

4.2 Kliens oldal



4.1. ábra: Kliensoldal folyamatainak bemutatása.

4.2.1 Keretrendszer bemutatása

Sveltekit

A SvelteKit egy modern webes keretrendszer, amely gyorsaságával, hatékony működésével és fejlesztőbarát megoldásaival emelkedik ki, különösen a statikus és dinamikus oldalak, valamint az egyszerű API-integráció támogatásában. Minimalizált kódméretének és beépített szerveroldali renderelésének (SSR) köszönhetően kiváló teljesítményt biztosít, így ideális alapot nyújt a szakdolgozatomban ismertetett alkalmazás frontendjéhez. [59]

4.2.2 Webfelületek és funkciók

Feltöltési felület

4.2. ábra: A feltöltési felület képernyőképe.

Az ábrán (képernyőképen) a program webes felületének kezdőoldala látható, amely az összefoglalási folyamat elindításához szükséges dokumentumok és médiatartalmak feltöltésére szolgál. A rendszer kizárólag a támogatott fájlformátumok feltöltését engedélyezi, ezáltal minimalizálva a feltöltési hibák lehetőségét és biztosítva a zökkenőmentes működést. A dokumentumok kiválasztása után a felhasználó személyre szabhatja az összefoglalás fogalmazási stílusát, amely a közérthető formától egészen a szakértői nyelvezetig terjed. Továbbá a biztonsági szempontokat mérlegelve elérhető egy lokális feldolgozási opció is, amely lehetővé teszi, hogy az összes feldolgozási művelet kizárólag a felhasználó számítógépén történjen, távoli szerverek igénybevétele nélkül.

A „Feltöltés” gomb aktiválásával a program a felhasználó által feltöltött dokumentumokat két különálló feldolgozási irányba továbbítja. Az egyik irány a Backend rendszer, ahol az összetettebb számítási műveletek zajlanak. Ebben a folyamatban a fájlok ideiglenesen egy lokális tárolóba kerülnek, amíg a rendszer kinyeri belőlük a szükséges információkat, például a fájlneveket és tartalmakat. A dokumentumok POST kérések formájában jutnak el a Backend-hez, ahol a sikeres feldolgozás eredményeként egy megfelelőségi címkével ellátva kerülnek vissza. A másik irány a felhasználói felületen való továbbítás és későbbi felhasználás, mint például a kommunikációs felületen való metaadatok kijelzéséhez.

A webfelület TypeScript részben helyet kap néhány funkció, mely továbbiakban fontos lesz a rendszer működés tekintetében.

Fájl metaadatainak eltárolása

```
1. function handleFileUpload(event: Event) {
2.   const target = event.target as HTMLInputElement;
3.   const files = target.files;
4.   if (files) {
5.     uploadedFiles = Array.from(files);
6.     isButtonDisabled = uploadedFiles.length === 0;
7.   }
8.     clearUploadedFiles();
9.     uploadedFiles.forEach(file => {
10.      const fileSize = file.size < 1024
11.        ? `${file.size} Bytes`
12.        : file.size < 1024 * 1024
13.        ? `${(file.size / 1024).toFixed(2)} KB`
14.        : file.size < 1024 * 1024 * 1024
15.        ? `${(file.size / (1024 * 1024)).toFixed(2)} MB`
16.        : `${(file.size / (1024 * 1024 * 1024)).toFixed(2)} GB`;
17.      const fileString = `${file.name}: (${fileSize})`;
18.      console.log("Hozzáadott fájl string:", fileString);
19.      addUploadedFile(fileString);
20.    });
21.    console.log("Store tartalma feltöltés után:", $uploadedFilesStore);
22. }
```

4.3. ábra: Fájlok feltöltése és azt követően a metaadataik mentése.

A függvény egy eseménytípusú paramétert fogad bemenetként, amely a fájlfeltöltési folyamatot reprezentálja, és kimenetként a feltöltött fájlok metaadatait tárolja egy helyi mappában.

A kód első lépésként az esemény fókuszát HTMLInputElement típusra alakítja, hogy hozzáférhessen a feltöltött fájlok tulajdonságaihoz. Ezt követően megpróbálja összegyűjteni a fájlokat. Ha ez sikeres, létrehoz egy tömböt, amelyben a fájlok adatai tárolódnak, és aktiválja a feltöltési gombot, lehetővé téve annak használatát.

A továbbiakban a kód alaphelyzetbe állítja a helyi tárolót, a fájlméreteket könnyen olvasható formátumra alakítja, majd az ekkor már üres tárolóba menti a fájlok metaadatait. Ennek célja, hogy a későbbi kommunikációs felületen a rendszer bemutathassa a felhasználónak az adatok forrását és relevanciáját.

```
async function SaveAllData() {
  const formData = new FormData();
  formData.append("isStoreInLocal", String(isStoreInLocal));
  formData.append("chatbotStyle", chatbotStyle);
  uploadedFiles.forEach((file) => {formData.append("files", file);});
  try {
    const response = await fetch('http://localhost:8000/api/saveAllData', {
      method: 'POST',
      body: formData,
    });
    if (!response.ok) {
      console.error("Hiba a metaadatok elküldésekor:", response.status);
      return;
    }
    window.location.href = "/loading";
  } catch (error) {
    console.error("Hálózati hiba történt:", error);
  }
}
```

4.4. ábra: Fájlok mentésének kérése a szerveroldaltól.

A bemutatott függvény a felhasználótól származó adatok logikai réteghez történő továbbításáért felel. Ez a művelet POST kérésen keresztül valósul meg, ahol a kérés törzsét a speciálisan definiált FormData képezi, amely tartalmazza a felület adatait. A FormData csomag elemei között szerepelnek a fájlok, a megadott fogalmazási stílus, valamint a feldolgozási módszer paraméterei.

Várakoztató felület

4.5. ábra: A várakoztató felület képrernyőképe.

Az ábrán a program webes felületének egy köztes állapota jelenik meg, amely valós idejű visszajelzést biztosít a felhasználók számára az éppen zajló folyamat aktuális szakaszáról. A funkció megvalósítása egy JavaScript függvény segítségével történik, amely egy API végpontról kér le adatokat csomagban.

```
1. import { onMount } from 'svelte';
2. onMount(() => {
3.     fetchMilestones();
4.     processFiles();
5.     const interval = setInterval(fetchMilestones, 3000);
6.     return () => clearInterval(interval);
7. });
```

4.6. ábra: Speciális életciklus a fájlok feldolgozásához.

Az ábrán bemutatott kód egy speciális életciklus-módszert implementál, amely lépésről lépésre hajt végre aszinkron hívásokat. Először a mérföldkövek létrehozását végzi el, majd kérést küld a logikai réteg felé a feltöltött állományok összefoglalására.

A processFiles (fájlok összefoglalása) folyamat állapotát a rendszer folyamatosan figyeli, három másodpercenként frissítve az információkat. Ez a megoldás hatékony valós idejű visszajelzést nyújt, mivel az időintervallum optimálisan van megválasztva: elég rövid ahhoz, hogy a nagyobb számítási igényű feladatok végrehajtási idejére ne gyakoroljon jelentős torzító hatást, ugyanakkor kellően hosszú a rendszer stabilitásának fenntartásához, minimalizálva az erőforrások túlzott igénybevételét és a túlterhelés kockázatát. A műveletek sikeres befejezését a rendszer vizuálisan, zöld pipával jelzi, és az összes lépés teljesítése után automatikusan átirányítja a felhasználót a következő felületre. Az új felület összegzi és bemutatja a háttérben végzett feldolgozás eredményeit, ezzel biztosítva a folyamatok átláthatóságát és könnyű nyomon követhetőségét.

```
1. type Milestone = {
2.   id: number;
3.   title: string;
4.   completed: boolean;
5. };
6. let milestones: Milestone[] = [];
7.
8. async function fetchMilestones() {
9.   try {
10.      const response = await fetch("http://localhost:8000/milestones");
11.      if (response.ok) {
12.          milestones = await response.json();
13.      } else {
14.          console.error("Hiba a mérföldkövek lekérdezésekor:", response.status);
15.      }
16.  } catch (error) {
17.      console.error("API hiba:", error);
18.  }
19. }
```

4.7. ábra: Mérföldkövek megvalósítása.

Az ábrán bemutatott függvény az előzőekben említett módon lekért adatokat használja fel az egyedileg definiált Milestone (mérföldkő) típusú változó feltöltésére. Ez a változó

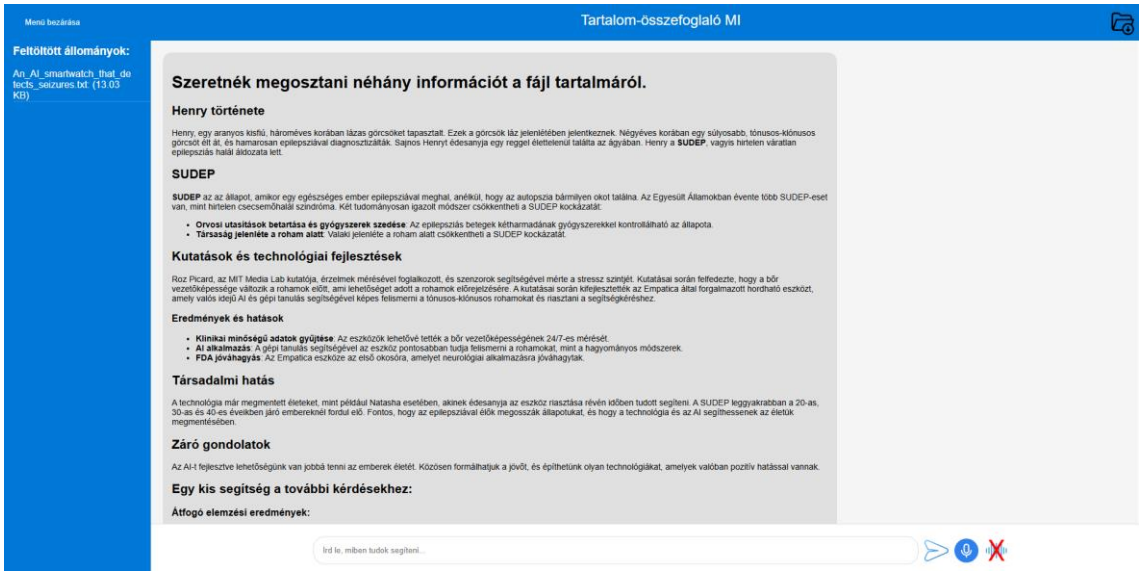
tartalmazza a feladat sorszámát, nevét és teljesítési státuszát. A változó frissítése egyúttal a webfelület vizuális megjelenítését (zöld pipa kihelyezését) is frissíti, ezáltal a felhasználók valós időben követhetik a változásokat.

A függvényeket aszinkron módon valósítottam meg, hogy egymás működését ne akadályozzák, és a technológia lehetőségeihez mérten a leghatékonyabb és leggyorsabb módon hajtsák végre a feladatokat.

A szöveges tartalom a folyamat során visszakerül a webes felületre, ahol az eltárolódik, és a POST kérések ezzel befejezik működésüket. Az ezt követő lépésként a szöveges tartalmat át kell helyezni a megnyíló webes felületre. Ennek megvalósítása érdekében a SvelteKit belső mappastruktúráját alkalmaztam, ahol nem fájl formájában tárolódik az adat, hanem a szükséges változókon keresztül kerül továbbításra a célterületre.

A folyamatok lezárását követően a felhasználó automatikusan a következő felületre navigálódik, ahol a következő munkafázis indul.

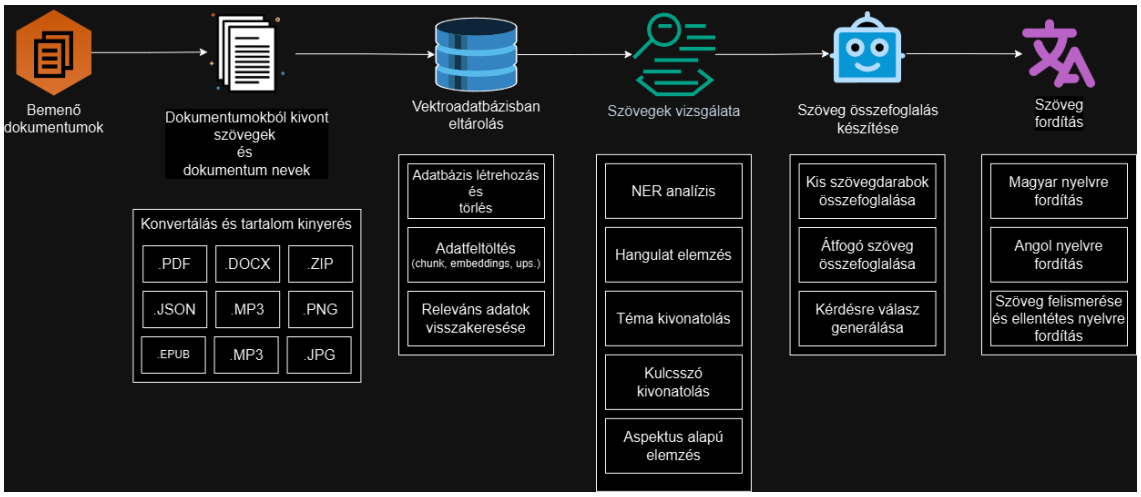
Kommunikációs felület



4.8. ábra: A kommunikációs felület képernyőképe.

A képernyőkép a mesterséges intelligencia és a felhasználó közötti reakciókat jeleníti meg. A fő felületen látszanak a modell válaszai (szürke buborék) és a későbbiekben megfogalmazódott felhasználó kérdések (kék buborék). Az alsó szekcióban a felhasználó szóban és írásban egyaránt kifejezheti reakcióit, illetve dönthet arról, hogy a rendszer válasza szöveges formában jelenjen meg, vagy aktiválja a válasz felolvasásának lehetőségét. A képernyő bal oldalán találhatóak a feltöltött fájlok metaadatai, ellenőrzési céljából. A jobb felső sarkában pedig lehetőség nyílik az egész beszélgetést JSON formátumban menteni a következő beszélgetés memóriájához.

4.3 Szerver oldal



4.9. ábra: Szövegösszefoglalás folyamatainak bemutatása.

4.3.1 FastAPI

A FastAPI-t azért választottam a szakdolgozatomhoz, mert modern, nagy teljesítményű, aszinkron működésű és skálázható keretrendszer, amely ideális mesterséges intelligencia alapú rendszerek fejlesztéséhez.

A szerveroldali fejlesztés és a vizuális oldal összekötése

```
1. from fastapi import FastAPI
2. import os
3. from fastapi.middleware.cors  import CORSMiddleware
4. app = FastAPI()
5. origins = ["http://localhost:5173"]
6. app.add_middleware(
7.             CORSMiddleware,
8.             allow_origins=origins,
9.             allow_credentials=True,
10.            allow_methods=["*"],
11.            allow_headers=["*"],)
```

4.10. ábra: Kliens és szerveroldal összekapcsolása.

A CORS lehetővé teszi, hogy egy másik domainről vagy originből érkező kérések elérhessék az API-t.

Az indulás első pillanatai

```
(venv) L:\Thesis\Backend>uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['L:\\Thesis\\Backend']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [25912] using WatchFiles
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Ben\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Ben\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Ben\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
INFO:     Started server process [25640]
INFO:     Waiting for application startup.
```

4.11. ábra: Backend indulása

A következő folyamatok láthatóak az ábrán:

- A projekt egy dedikált virtuális környezetben fut, amely garantálja a megfelelő izolációt és kompatibilitást.
- A kimenet igazolja, hogy a Uvicorn szerver sikeresen elindult, és a webalkalmazás elérhető a 127.0.0.1:8000 címen.
- Az újratöltési funkció aktív, lehetővé téve, hogy a rendszer automatikusan újrainduljon a fájlok módosításakor.
- Az nltk csomagok frissítésre kerülnek, biztosítva a szövegelemzési funkciók naprakész és teljesen lokális környezetben történő használatát.
- A rendszer visszajelzést ad, hogy az alkalmazás készen áll a használatra.

4.3.2 Adatfeldolgozási módszerek mérlegelése

A legcélravezetőbb megközelítés a lokális és globális AI modellek lehetőségének együttes biztosítása, hogy a felhasználók igényeik és körülményeik szerint választhassanak. A lokális modellek különösen az adatvédelem és az internetfüggetlenség szempontjából jelenthetnek előnyt, míg a globális modellek a rugalmasságot és skálázhatóságot helyezik előtérbe. E kettős lehetőség biztosítja a legjobb megoldás kiválasztását az eltérő felhasználói elvárások alapján.

Láncolatokról

A LangChain keretrendszer számos előnyének köszönhetően modern és jól strukturált fejlesztési környezetet biztosít. Az egyik legfőbb előnye, hogy láncolatai révén a rendszer folyamatai átláthatók, ami nagymértékben megkönnyíti a későbbi fejlesztéseket.

A globális és lokális irányok hatékony kezelése érdekében külön modult hoztam létre, amely egyszerű és átlátható működést tesz lehetővé. A láncolatok alapvetően előre definiált láncszemekből állnak, amelyek a mikroszolgáltatásokat reprezentálják, és szekvenciális folyamatokká kapcsolódnak össze. Ezáltal az egyik láncszem kimenete közvetlenül a következő bemenetéül szolgál, biztosítva a gördülékeny és logikus működést.

```
load_files_chain = LoadFilesChain()
vector_store_chain = VectorStoreChain(vector_size=1536)
analyze_process_chain = AnalyzeProcessChain()
summarize_content_chain = SummarizeContentChain()

global_summarizer_chain = SequentialChain(
    chains=[load_files_chain, vector_store_chain, analyze_process_chain, summarize_content_chain],
    input_variables=["file_path", "stored_data"],
    output_variables=["final_summary"]
)
```

4.12. ábra: Szekvenciális láncolat szemléltetése.

A kód az egyes láncszemekből felépített láncolatot mutatja. A továbbiakban egyesével kifejtem a láncszemek működését.

4.3.3 Fájl konvertáló modul

Célja a feltöltött fájlok tartalmának kinyerése és szótár alakban visszaadni őket a következő formában: kulcs(fájlnév) - érték(tartalom). Ez a következőképpen valósul meg:

```
1. def process_files_in_directory(file_path:  str) -> Dict[str, str]:
2.    if not os.path.isdir(file_path):
3.       raise FileNotFoundError(f"A megadott könyvtár nem található:  {file_path}")
4.
5.    file_contents  = {}
6.    for filename  in os.listdir(file_path):
7.       full_path = os.path.join(file_path,  filename)
8.       if os.path.isfile(full_path):
9.          try:
10.             content = extract_text_from_file_sync(full_path)
11.             file_contents[filename]  = content
12.          except Exception  as e:
13.             print(f"Hiba történt a {filename} fájl feldolgozása során: {str(e)}")
14.       shutil.rmtree(file_path)
15.    return file_contents
```

4.13. ábra: Fájlok tartalmának kigyűjtése.

A feltöltési felület POST kérése során a szerveroldal egy ideiglenes helyi mappa létrehozását kezdeményezi a fájlok tárolására. A fájltartalom betöltésére szolgáló láncszem lekéri ennek a mappának az elérési útvonalát, majd egyenként feldolgozza a fájlokat, azok tartalmát szöveges formátumban kinyerve.

A fájlok beolvasása után, az adatbiztonság érdekében a rendszer azonnal törli a már feldolgozott fájlokat. A folyamat lezárásaként elkészül a korábban említett szótár, és az ideiglenes mappa automatikusan megszűnik, biztosítva a tiszta és biztonságos működést.

4.3.4 Szöveg tároló modul

A fájlok tartalmának hatékony keresését nagymértékben javítja, ha a szövegeket kisebb részekre bontjuk, és ezekhez vektorokat rendelünk. Ez lehetővé teszi, hogy cosinus-távolság metrika alapján hasonlóságot állapítsunk meg az egyes szövegelemek között. A koszinusz távolság képlete a következő

Az eljárás az alábbi lépésekből áll:

1. Adatbázis alaphelyzetbe állítása: Az előző futásból származó zajok kiküszöbölése érdekében a gyűjteményt teljesen törli a rendszer.
2. Új adatok betöltése az üres tárolóba:

- A szöveget kisebb részekre bontja szemantikai módszerrel, amely a darabolást nem csupán adott karakterenként végez, hanem szöveg értelem tekintetében is.
- Minden szövegrészhez vektort rendel egy modell segítségével. Ehhez globális (pl. OpenAIEmbeddings) és lokális (pl. sentence-transformers/all-MiniLM-L6-v2) megoldásokat alkalmaz.
- Az adatbázisba a következőket tárolja el:
    - Egyedi azonosítót, nagyobb mennyiségű indexelés érdekében.
    - Vektort, amely a modell által számított reprezentációja a szövegrészletnek.
    - Tartalmat, amely tartalmazza a fájl nevét és a szövegrészletet.

A modul tartalmaz egy függvényt, amely a felhasználó kérdését vektorizálja, majd megkeresi az öt legjobban illeszkedő szövegrészt. Ezeket továbbítja egy nagy nyelvi modellnek, amely a lehető legrelevánsabb választ adja a kérdésre.

A gyorsabb válaszadás érdekében a modul a csevegési reakciókat is eltárolja. Így a korábban megválaszolt kérdések esetében a rendszer még hatékonyabban tudja biztosítani a helyes válaszokat.

4.3.5 Szöveg analizáló modul

A modul célja, hogy átfogó átvilágítást tegyen a beérkező fájlok tartalmából. Lokális megvalósításban több oldalról vizsgálja a szövegeket, ami végül a felhasználónak egy segítséget ad még átfogóbban látni a tartalmat. Az egyes folyamatokat párhuzamosan futtatja a rendszer, ami azt is eredményezheti, hogy különböző komplexitású szöveg eredményeit eltérő sorrendben jelenít meg. A következő analízisek történnek meg egy futás alatt:

Entitás felismerés

A szöveg átvilágítását a „spacy("en_core_web_sm")” modell segítségével végeztem, amely egy gyors és könnyű modell direkt erre a feladatra kifejlesztve. A modell egy dokumentumot vár majd ezt követően címkéket helyez a szövegbe kategóriájának megfelelően. A művelet után a program entitásokon lépkedve listába gyűjti az azonos kategóriájú elemeket melyek a következőek:

(Személyek), (Nemzeti, vallási vagy politikai csoportok), (Épületek, létesítmények), (Szervezetek, cégek, intézmények), (Földrajzi-politikai entitások), (Helyek, lokációk), (Termékek, tárgyak), (Események), (Jogszabályok, törvények nevei), (Műalkotások), (Nyelvek), (Időhöz kapcsolódó kifejezések), (Pénzösszegek), (Sorszámok)

Hangulat elemzés

A függvény célja meghatározni egy szövegben felmerülő érzelmek automatikus felismerését. A kimenet háromféle értéket vehet fel, mely a pozitív (jó), negatív (szomorú) és egy köztes állapot, amit a rendszer vegyesnek bélyegez.

A szöveg érzelmi átvizsgálásához a Huggingface továbbfejlesztett "distilbert-base-uncased-finetuned-sst-2-english" modelljét használtam, mert könnyű és gyors DisilBERT alapú modell direkt erre a célra finomhangolva. Azonban a modell nem képes hatalmas szövegek egyszerre történő elemzésére, így kézenfekvő megoldásnak számít kisebb részekre bontani és végül átlagolni a kis részegységek eredményeit egy átfogó eredménnyé.

Aspektus alapú véleménybányászat

A függvény azt vizsgálja, hogy a szöveg mennyire kapcsolódik az előre definiált témákhoz. Ennek megvalósításához a "facebook/bart-large-mnli" modelljét használtam, ami kellően nagy mennyiségű tanítóadattal rendelkezik, ahhoz, hogy az eredmény valósághű legyen.

A szöveget kiértékeli az előre megadott aspektusok alapján, és minden aspektushoz társít egy relevancia-pontszámot. Majd ezekből eldönti, hogy mennyire volt hangsúlyos.

Téma kivonatolás

Ez a függvény a megadott szövegből, először eltávolítja a „stopwords” vagyis magyarosan töltelékszavakat és a maradék szövegben témákat azonosít Látens Dirichlet Analízis segítségével. Ez a statisztikai eljárás arra a célra szolgál, hogy a szövegek szavaiból a rejtett (lappangó) témák valószínűségi mintázatait modellezze. A töltelékszavak azonosítására nltk corpus-t használtam, ugyanis nagyon pontosan definiált töltelékszó listából dolgozik többféle nyelven. A statisztikai modellhez pedig a gensim könyvtárt hívtam segítségül a memóriahatékonysága és az optimalizált teljesítménye miatt, akár hatalmas méretű adathalmazokon is.

Kulcsszavak kivonatolása

A függvény megadott számú kulcsszót emel ki a szövegből kontextus alapján. A kulcszavak azonosításához „keybert” könyvtárat használtam, mivel ez a modell nem csupán statisztikai gyakoriság alapján hozza meg a döntését, hanem valódi jelentést tükröz az eredménye. Emellett rendkívül gyors és támogatja az összetett szavas kulcsszavakat.

4.3.6 Szöveg összefoglaló modul

A szöveg összefoglalása ismét két irányra oszlik, figyelembe véve a feldolgozás módszerét:

Globális szöveg összefoglalás

A szöveggenerálási feladathoz ezen szintéren OpenAI GPT-4o modellt használtam a megbízhatósága és egyszerű integrációja miatt. Az modell egy API kulcs segítségével működtethető, amelyet a program gyökérmappájának „.env” fájljában tároltam el.

A szöveg feldolgozási feladatok alatt két féle nehézség adódik:

- A bemeneti tokenszám (szövegegység szám) korlátozottsága: A modell bemeneti tokenszáma 32768, ami azt jelenti, hogy ezt a számot nem haladhatja meg az összefoglalandó szövegdarab, a prompt és a generálásra kért tokenek számának együttes összege. Ha túl nagy a három összetevőből valamelyik, akkor gyakran hibára futhat a generálás.
    - Megoldásként a szöveg kis darabokra szeletelését választottam és párhuzamos módon való feldolgozását alkalmaztam.
- Az egy perc alatt lekérhető tokenszám: A modell meghatározza, hogy egy perc alatt mennyi tokent kérhet le a rendszer, ami a párhuzamos feldolgozás végett nagyon megugrik.
    - Megoldásként egy védelmi funkciót építettem be, hogy ha ilyesfajta túllépés következik be, akkor várakoztassa a generálást amig szükséges.

```
1. async def generate_summary(self, documents: dict[str, str], sum_style: str, collection_name: str) -> str:
2.     try:
3.         summary = ""
4.         prompt_complex = self.promptGenerator.create_long_summary_prompt_hu(sum_style)
5.         prompt_simple = self.promptGenerator.create_short_summary_prompt_hu(sum_style)
6.         prompt_token_count = self.get_prompt_token_count(prompt_simple)
7.
8.         max_token_numb = self.max_model_tokens - prompt_token_count - 500
9.
10.        tasks = [
11.            self.process_document(doc_name, doc_content, prompt_simple, max_token_numb)
12.            for doc_name, doc_content in documents.items()
13.        ]
14.
15.        summaries = await asyncio.gather(*tasks)
16.
17.        for part_of_summary in zip(documents.keys(), summaries):
18.            response = await self.call_openai_api(prompt_complex, part_of_summary, 2048, 0.2)
19.            summary += response + "\n"
20.
21.        store_string_to_qdrant(summary,"bot",collection_name)
22.
23.        return summary
24.    except Exception as e:
25.        logging.error(f"Hiba történt a dokumentum összefoglalás során: {e}")
26.        return ""
```

Promptok átadása fogalmazási stílus szerint.

Maximális tokenszám kalkuláció.

Kis fogalmazások megiratása párhuzamos módon.

Átfogó fogalmazás készítése.

Összefoglalás eltárolása a vektoradatbázis-ban.

4.14. ábra: Tokenszám túllépés kiküszöbölése.

A kód a bemeneti tokenszám problémáját dolgozza fel, melyben egy előre kiszámolt maximális tokenszám alapján pontosan akkora tokenszám hosszúságúra darabolja a szöveget, hogy a későbbiekben ne okozhasson korlátozottsági hibát a limit elérése.

```python
async def call_openai_api(self,  prompt: str, chunk: str, max_tokens: int, temp: float, max_retries: int = 5) -> str:
    retry_count = 0
    while retry_count < max_retries:
        try:
            response = await asyncio.to_thread(
                openai.ChatCompletion.create,
                model=self.model,
                messages=[
                    {"role": "system", "content": "You are an expert assistant summarizing documents."},
                    {"role": "user", "content": prompt + chunk}
                ],
                max_tokens=max_tokens,
                temperature=temp
            )
            return response['choices'][0]['message']['content']
        except openai.error.RateLimitError as e:
            wait_time = float(str(e).split("try again in")[1].split("s")[0].strip())
            print(f"Rate limit elérve, várakozás {wait_time} másodpercig...")
            await asyncio.sleep(wait_time)
            retry_count += 1
        except Exception as e:
            print(f"Hiba történt az OpenAI API hívás során: {e}")
            break
    print("Az újrapróbálkozások száma elérte a maximumot.")
    return ""
```

4.15. ábra: "Token / Perc" kiküszöbölése az aszinkron szöveggenerálások során.

A kód a TPM hibát dolgozza fel, melyben újrapróbálkozásokat eszközöl az egy perc alatt lekérhető tokenek elérése után. A kivételkezelésben lévő hiba szövegéből kiolvassa az időpontot, amikor újra generálhat szöveget és addig várakozik.

A függvény egy nagyon ritka kimenetelén a próbálkozások száma elérheti az előre definiált maximum értéket, ekkor az erőforrások felszabadítása és a gyorsaság érdekében kompromisszumot kell kötnünk és újabb szöveg darabra térni, későbbiekben van lehetőség az információk pontosítására.

Lokális szöveg összefoglalás

A szöveggenerálási feladatokhoz "meta-llama/Llama-3.2-1B" modellt alkalmaztam, mert egy milliárd paraméteres méretének köszönhetően kevesebb számítási erőforrást igényel, így alkalmas helyi eszközön történő futtatásra is. Ez a modell kifejezetten párbeszédes alkalmazásokhoz lett optimalizálva, beleértve a szöveg összefoglalást és a kérdés válasz feladatokat is. A modell nyílt forráskodú, azonban a Huggingface weboldalról kötelező kulcsot igényelni a használatához, mely ingyenes szolgáltatás. Ezt szintén az „.env”-ben tárolom.

A fejlesztés során hasonló módon megjelent a tokenszám túllépés hibája (mely jelen esetben még kevesebb token számot enged meg), melynek megoldásánál kifejezetten ügyeltem arra, hogy a megoldásban hasonló felépítésben oldja meg a problémát, mint a globális módszernél, így elősegítve a könnyebb fejlesztést a jövőben.

A lokális megvalósítás egyik előnye, hogy az adatokat saját eszközön tudjuk feldolgozni, melynek kétféle módja van:

- CPU futtatás: A lassabb megoldás, mert a CPU általában kevesebb maggal rendelkezik és ezek inkább soros feldolgozásra optimalizáltak, nem pedig párhuzamos feladatok végrehajtására.
- GPU futtatás: A gyorsabb megoldás, mert a GPU nagy számú maggal rendelkezik, amelyek egyszerre párhuzamosan képesek működni.

```
1. import torch
2. self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

4.16. ábra: GPU feldolgozást beállító elágazás.

A kód megmutatja a módszert, ahogyan ellenőrzi a számítógép grafikus feldolgozó egységének torch támogatását. Ha elérhető a többnyire NVIDIA eszközöket támogatott cuda programozási modell, akkor a fejlesztő képes a gyorsabb lehetőséget kihasználva futtatni a modellt.

```
1. input_ids = self.tokenizer(prompt, return_tensors="pt").input_ids.to(self.device)
2. if self.tokenizer.pad_token_id is None:
3.     self.tokenizer.pad_token_id = self.tokenizer.eos_token_id
4.
5.         with torch.no_grad():
6.             output = self.model.generate(
7.                 input_ids=input_ids,
8.                 attention_mask=torch.ones_like(input_ids).to(self.device),
9.                 max_new_tokens=150,
10.                 do_sample=True,
11.                 temperature=0.7,
12.                 repetition_penalty=1.2,
13.                 pad_token_id=self.tokenizer.pad_token_id
14.             )
```

4.17. ábra: Llama modell szöveggenerálási struktúrája.

A kód részlet bemutatja, hogy az előző beállítások után, hogyan generál egy adott szöveget. Eleinte az input_ids változóba tokenizáljuk a rendszer instrukcióit és Pytorch Tensor formára hozva, mely egy olyan mátrix, amelyet könnyen lehet továbbítani a neurális hálózatokba. Továbbá egy vizsgálatot végez a helykitöltővel kapcsolatban, ez gyakran olyan speciális token, amely a szöveg befejezését jelenti.

A végső block elvégzi a szöveggenerálást a következőképpen:

- input_ids: a bemeneti utasításokat tartalmazza. (pl: összefoglalási kérelem)
- attention_mask: az input szekvenciában mely tokenekre kell figyelmet fordítani, és melyeket kell figyelmen kívül hagyni.
- max_new_tokens: korlátozza a generált válasz hosszát.
- do_sample: modell sztochasztikus mintavételt alkalmaz, a következő token kiválasztása valószínűségi eloszlásból történik.
- temperature: kreativitás beállítása.

- repetition_penalty: az ismétlődést büntető paraméter.
- pad_token_id: helykitöltő.

A modellben továbbá különbséget jelent, hogy bár támogatja a többnyelvűséget, sajnos a magyar nyelv nem került bele ebbe a palettába, így egy fordító ágenst kellett eszközölni, ami megpróbálja a lehető legmagyarosabb módon visszaadni a tartalmat.

```
1. from transformers import pipeline, AutoTokenizer
2.
3. self.en_to_hu_translator = pipeline("translation_en_to_hu", model="Helsinki-NLP/opus-mt-en-hu")
4. self.hu_to_en_translator = pipeline("translation_hu_to_en", model="Helsinki-NLP/opus-mt-hu-en")
5. self.en_to_hu_tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/opus-mt-en-hu")
6. self.hu_to_en_tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/opus-mt-hu-en")
```

4.18. ábra: Fordító modellek és tokenizálók betöltése.

A nyelvfordítási problémára a Helsinki-NLP Opus-MT modelljeit alkalmaztam, mivel ezek specializált, előre betanított fordítómodellek angol-magyar és magyar-angol nyelvpárokra. A pipeline API-t használtam, amely egyszerűsíti a fordítási feladatok integrációját, és egyben automatikusan kezeli a tokenizálást, dekódolást, valamint a modell működését. A tokenizálók külön inicializálása lehetővé teszi a finomhangolást vagy a tokenizálás testreszabását, ha szükséges, miközben az alapértelmezett beállításokat is támogatja a gördülékeny működés érdekében.

4.3.7 Szöveg visszakereső funkcó

A szöveg visszakereső funkció mind globális mind lokális megvalósításban szintén hasonlóan működik a fordítási feladattól eltekintve.

A nyelvmodell paraméterében megkapja a felhasználó kérdését és a vektoradatbázis gyűjtemény nevét.

1. Az öt legrelevánsabb információ megkeresése a vektoradatbázisban, amely legjobban illeszkedik a kérdés vektorjához.
2. Eltárolja a kérdést a vektoradatbázisban, hogy továbbiakban a hasonló kérdést még gyorsabban és pontosabban meg tudja válaszolni.
3. A releváns adatokból generál egy szöveget, amely a kérdésre irányul.
4. Eltárolja a kérdésre tett választ.
5. Visszaküldi a POST kérés válaszát, amely a webfelületre kiírásra kerül.

4.3.8 Program konténerizálása

Mivel a projectem két fő részből áll a szerver oldal (Python alapú) és a kliensoldal (SvelteKit alapú), különálló Docker image (képeket) kell készíteni mindkét komponens számára, majd egy Docker Compose segítségével futtatni őket együttesen.

A következő lépésekben bemutatom a konténerizálás menetét:

```
FROM python:3.10-slim

WORKDIR /app

RUN apt-get update && apt-get install -y \
    libglib2.0-0 \
    libsm6 \
    libxrender-dev \
    libxext6 \
    ffmpeg \
    tesseract-ocr \
    && rm -rf /var/lib/apt/lists/*

COPY . /app

RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt

EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

4.19. ábra: Szerveroldali függőségek konténerizálása.

Első lépésként elkészítettem a szerveroldal Docker fájlját, amely beállítja a rendszert, telepíti a függőségeket és futtatja a programot.

```
FROM node:18

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .
EXPOSE 5173
CMD ["npm", "run", "dev", "--", "--host"]
```

4.20. ábra: Kliensoldali függőségek konténerizálása.

Második lépésként létrehoztam a kliensoldal Docker fájlját, amely beállítja a rendszert, másolja az alapcsomagokat, és elindítja a webfelületet a megfelelő porton.

A harmadik lépésben a Docker szolgáltatásokat egy Compose fájlba rendeztem, amely a build paranccsal felépíti a függőségeket és az up paranccsal indítja a programot.

```
version: '3.8'
services:
  backend:
    build:
      context: ./Backend
    ports:
      - "8000:8000"
    volumes:
      - ./Backend:/app
    environment:
      - PYTHONUNBUFFERED=1
  frontend:
    build:
      context: ./Frontend/Sveltekit
    ports:
      - "5173:5173"
    volumes:
      - ./Frontend/Sveltekit:/app
    command: ["npm", "run", "dev", "--", "--host"]
```

4.21. ábra: Szerver és kliensoldal összeállítása.

5 TESZTELÉS

5.1 Kliensoldal tesztelése

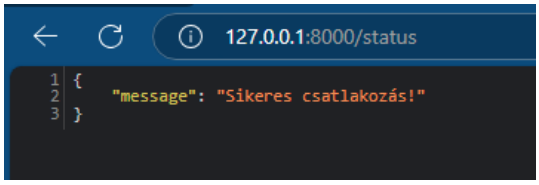
A kliens oldalt manuálisan teszteltem, majd az eredmények alapján levontam a szükséges következtetéseket.

Feltöltési felület megjelenítése.	sikeres ✓
↳ Adatok mentése.	sikeres ✓
Várakoztató felület megjelenítése.	sikeres ✓
↳ Szerver állapot kijelzése.	sikeres ✓
Kommunikációs felület megjelenítése.	sikeres ✓
↳ Fájlok adatainak kijelzése a menüben.	sikeres ✓
↳ Modell reakcióinak jelzése és formázása.	sikeres ✓
↳ Szövegbevitel és buborék létrehozás.	sikeres ✓
↳ Hang alapú szövegírás.	sikeres ✓
↳ Szöveg felolvasása automatikusan.	sikeres ✓
↳ JSON fájl mentése a beszélgetésről.	sikeres ✓

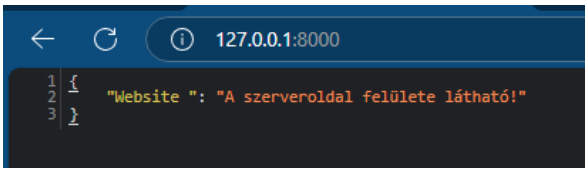
5.1. táblázat: A kliensoldali funkciók egységtesztje.

5.2 Szerveroldal tesztelése

A következő képernyőképek szemléltetik, hogy a csatlakozás sikeresen létrejött, és a kliens és a szerver oldal képes kommunikálni egymással.



5.1. ábra: Két oldal sikeres csatlakozás



5.2. ábra: Teszt: FastAPI működés

5.3. ábra: A szerveroldal eléri a fájlok metaadatát.

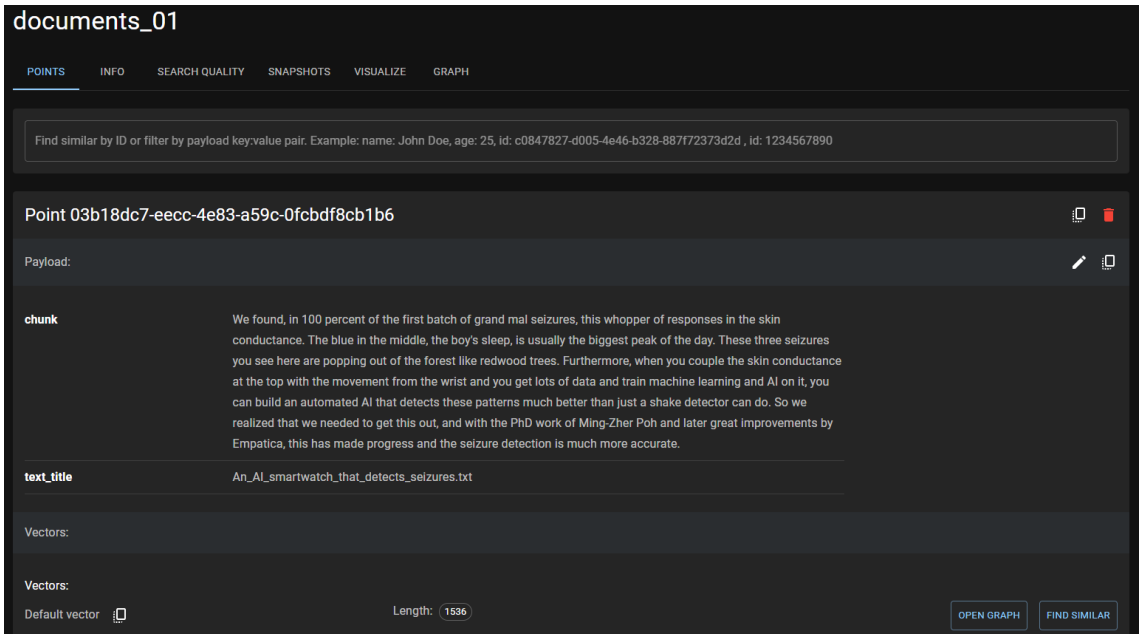
5.3 Tartalom átalakítás tesztelése

Ebben a tesztben minden elfogadható fájlkiterjesztésben feltöltöttem fájlokat.

’.docx’ formátum átalakítása folyószöveggé.	sikeres ✓
’.pdf’ formátum átalakítása folyószöveggé.	sikeres ✓
’.json’ formátum átalakítása folyószöveggé.	sikeres ✓
’.epub’ formátum átalakítása folyószöveggé.	sikeres ✓
’.png’ formátum átalakítása folyószöveggé.	sikeres ✓
’.jpg’ formátum átalakítása folyószöveggé.	sikeres ✓
’.mp3’ formátum átalakítása folyószöveggé.	sikeres ✓
’.mp4’ formátum átalakítása folyószöveggé.	sikeres ✓
’.zip’ fájljainak átalakítása folyószöveggé.	sikeres ✓

5.2. táblázat: Fájlformátumok beolvasási tesztje.

5.4 Vektoradatbázis tesztelése



5.4. ábra: Teszt: az újonnan létrejövő gyűjteményről

5.5 Szöveg analízis tesztelése

Az elemzés hitelességének ellenőrzéséhez egy előre ismert kimenetelű szöveget generáltam. Az azonosított entitások számát az összes esethez viszonyítva értékeltem.

Vizsgált szempont	Darabszám	Talált	Teljesítmény
Személyek	10	10	100 %
Nemzeti, vallási vagy politikai csoportok	10	6	60%

Szervezetek	10	9	90 %
Helyek	10	10	100 %
Események	10	7	70 %
Időhöz kapcsolódó kifejezések	10	9	90 %
Témák	Megfelelő témajavaslatot tett.		
Kulcsszavak	Megfelelő kulcsszavakat talált.		
Hangulat	Helyesen értékelte a szöveg hangulatát.		

5.3. táblázat: A szöveg analízis tesztelése véletlenszerű írott anyagon.

5.6 Szöveggenerálás tesztelése

A tartalom-összefoglalás minősége relatív fogalom, amelynek mérése jelentős kihívást jelent. Ennek érdekében két különböző módszert alkalmazok a végső értékelés megközelítésére.

Az összefoglalás lényegretörőségének mérése karakterszámok alapján:

Tartalom hossza (karakterben)	Globális összefoglalás hossza (karakterben)	Lokális összefoglalás hossza (karakterben)
7.310	1.065	5.064
10.441	1.231	7.248

5.4. táblázat: Összefoglalások hossza karakterben.

A szöveggenerálás minőségének értékeléséhez a Microsoft Copilot mesterséges intelligenciáját választottam, mivel célom egy független és objektív döntéshozatal megvalósítása volt. Bár a Copilot GPT-4 alapokra épül, úgy véltem, hogy a szakdolgozatban elemzett modellek és azok gyártói irányába semleges álláspontot képviselhet.

Fontosnak tartottam azonban bemutatni a modellek közötti lényeges különbségeket, különös tekintettel a paraméterszámokra. Az OpenAI GPT-4o pontos paraméterszáma jelenleg nem nyilvános, de elődje, a GPT-3 modell körülbelül 175 milliárd paraméterrel rendelkezett, így a negyedik generáció valószínűsíthetően ezt jelentősen meghaladja. Ezzel szemben a szakdolgozatban vizsgált másik modell, a Llama 3.2 1B, egy kisebb, 1 milliárd paramétert tartalmazó modell, amely specifikusan kisebb méretű és célzott felhasználásra optimalizált rendszert képvisel. [25] [60]

A döntéshozó mesterséges intelligencia értékelésének során ezen paraméterbeli különbségeket is figyelembe kellett vennie, hiszen a modellméret és a felhasználási kontextus nagyban befolyásolja az adott rendszerek teljesítményét és a generált szövegek minőségét. Ezen elemzésen keresztül biztosítottam, hogy a szakdolgozatom elemzése kiegyensúlyozott és tényszerű maradjon.

Szöveg összefoglalás	
Globális megvalósítás	Lokális megvalósítás
A válaszai pontosak és kellően tömörek, így gyorsan érthetők.	A válaszai informatívak, de ha egy kicsit tömörebben fogalmazna, még hatásosabbak lehetnének.

5.5. táblázat: Szövegösszefoglalás minősítése.

Kérdésre adott válasz	
Globális megvalósítás	Lokális megvalósítás
Megfelelő válasz, tömör információval.	Megfelelő válasz, bővebb információval.

5.6. táblázat: Kérdésre adott válasz minősítése.

5.7 Szöveg fordítás tesztelése

A nyelvfordító ágens működését két véletlenszerűen kiválasztott szöveg fordítása és az elkészült fordítások pontosságának alapos ellenőrzése alapján értékeltem.

```
INFO:     127.0.0.1:57159 - "GET /milestones HTTP/1.1" 200 OK
# Az angol nyelvű teszt szöveg:
TED Talks inspire people worldwide through powerful, concise, and insightful ideas.
# Az ágens fordítási eredménye angol nyelvről magyarra:
A TED Talks erőteljes, tömör és éleslátó ötletekkel inspirálja az embereket világszerte.
# Az magyar nyelvű teszt szöveg:
A TED-előadások innovatív ötleteket osztanak meg, amelyek kíváncsiságot ébresztenek és változást hoznak.
# Az ágens fordítási eredménye magyar nyelvről angolra:
TED lectures share innovative ideas that arouse curiosity and change.
INFO:     127.0.0.1:57222 - "GET /milestones HTTP/1.1" 200 OK
```

5.5. ábra: Teszt: az ágens fordítási eredményei

Az ágens fordítási eredményei megfelelőek a program rendeltetésszerű működéséhez.

6 TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A mesterséges intelligencia rendszerek fejlesztésében kulcsfontosságú, hogy a technológiák ne csak aktuális problémákat oldjanak meg, hanem folyamatosan fejlődjenek és új képességeket sajátítsanak el.

Finomhangolás: a személyre szabott teljesítmény kulcsa

A nagy nyelvi modellek finomhangolása lehetővé teszi, hogy kifejezetten célzott feladatok elvégzésére optimalizáljuk őket. A jelen szakdolgozat a TED Talks konferenciák anyagait dolgozza fel, amelyek rendkívül széleskörű témákat ölelnek fel. Ezért a modell eredeti, általános működésmódja kevésbé hatékony az ilyen diverz tartalmak feldolgozásában. A témakör szűkítése és a modell specifikus finomhangolása révén azonban rendkívül emberközeli és precíz eredményeket érhetünk el, tovább növelve a rendszer alkalmazhatóságát és relevanciáját.

Automatikus tanulási mechanizmusok: a folyamatos fejlődés záloga

Ez a szekció egy olyan fejlesztési ötletet tárgyal, amelynek célja a feldolgozási folyamatok mérhető javítása. Az elképzelés lényege, hogy a rendszer opcionális visszajelzést kér a felhasználótól, és az így kapott információk alapján finomítja a szöveggenerálás minőségét. Ezzel nemcsak a rendszer hatékonysága növelhető, hanem a felhasználói igények pontosabb kiszolgálása is biztosítható.

Fájlok tartalmának mélyebb módszerrel való kinyerése

Jelenleg a program nincs megfelelően felkészítve arra, hogy a dokumentumokban található szöveges adatokon túlmenően egyéb információforrásokat is figyelembe vegyen az információszerzés érdekében. Ennek következtében nem elemzi az alakzatokat, struktúrákat vagy egyéb releváns elemeket, amelyek szintén lényeges szerepet játszhatnak az információ hatékony kinyerésében. Az ilyen elemek figyelmen kívül hagyása korlátozhatja a rendszer sokoldalúságát és a kinyerhető adatok teljes spektrumát.

7 ÖSSZEFOGLALÁS

A szakdolgozat feladatom egy Tartalom-összefoglaló Mesterséges Intelligenciát tartalmaz, amely egy RAG microservice projectet testesít meg. A projekt korszerű fejlesztési technikákra épül, mint a modularitás, verziókövetés, konténerizáció, modern keretrendszerek használata és a decentralizált architektúra.

Az irodalomkutatás során részletesen bemutattam a nyelvfeldolgozó mesterséges intelligencia területének legfontosabb alaptémaköreit. Ezek közé tartozott a szöveg tisztításának módszertana, a szöveg különböző entitásainak azonosítása, a mesterséges hangképzés technikáinak ismertetése, az intelligens ágensek működésének bemutatása, a nagy nyelvi modellek átfogó áttekintése, valamint a modern fejlesztői keretrendszerek alkalmazhatósága és hordozhatóságuk elemzése. Az irodalomkutatás eredményeként egy szilárd elméleti alapot sikerült lefektetnem, amelyre támaszkodva megalapozottan kezdhettem el a rendszerterv kidolgozását.

A rendszerterv szekciójában részletesen bemutattam a szakdolgozatom feladatai megvalósításának tervét, amely szilárd alapokra épült az irodalomkutatás során megszerzett mélyreható elméleti ismeretek révén. A fejlesztési folyamat minden szakaszában szigorúan követtem az előre kidolgozott, részletes vázlatot, amely egyértelmű iránymutatásként szolgált. Ez a módszer lehetővé tette, hogy egy átlátható, strukturált és alaposan dokumentált szoftverterméket hozzak létre, amely maradéktalanul megfelel a kezdetben megfogalmazott terveknek és a projekt meghatározott célkitűzéseinek.

A tesztelési folyamat során egységtesztekkel ellenőriztem a program különálló részeinek helyes működését, különös figyelmet fordítva a funkcionalitás részleteire. Ezen túlmenően manuális ellenőrzési módszerekkel vizsgáltam a szövegtartalom összefoglalását, a szöveg entitásainak elemzését, valamint az adatok helyes és pontos tárolását. Emellett ellenőriztem, hogy a program által adott válaszok megfeleltek-e a kérdésekre, biztosítva a funkcionalitás pontosságát és a rendszer megbízhatóságát.

A szakdolgozat zárásaként a program továbbfejlesztésére irányuló innovatív javaslatokat ismertetek, amelyek a funkcionalitás bővítését és a hatékonyság növelését célozzák. Ezek közé tartozik a mesterséges intelligencia modellek finomhangolása és ezzel szorosan együttműködő automatikus tanulási mechanizmus bevezetése.

Összegzésként elmondható, hogy a bemutatott javaslatok nemcsak a program funkcionalitásának bővítését, hanem a hatékonyság jelentős növelését is elősegíthetik, ezzel hozzájárulva a rendszer jövőbeli versenyképességéhez és fenntartható fejlődéséhez.

8 SUMMARY

My thesis task involves a Content-Summary Artificial Intelligence that embodies a RAG microservice project. The project is built on modern development techniques, such as modularity, version control, containerization, the use of modern frameworks, and decentralized architecture.

During the literature review, I provided a detailed overview of the key foundational topics in the field of natural language processing artificial intelligence. These included methodologies for text cleaning, identification of various text entities, techniques for artificial speech generation, demonstration of the operation of intelligent agents, a comprehensive review of large language models, and an analysis of the applicability and portability of modern development frameworks. As a result of the literature review, I established a solid theoretical foundation, enabling me to commence the design of the system in a well-informed manner.

In the system design section, I presented the detailed plan for implementing the tasks outlined in my thesis, which was grounded on the in-depth theoretical knowledge acquired during the literature review. Throughout every phase of the development process, I strictly adhered to the pre-developed, detailed outline, which served as a clear guideline. This approach allowed me to create a transparent, structured, and thoroughly documented software product that fully met the initial plans and the project's defined objectives.

During the testing process, I verified the correct functioning of individual components of the program through unit tests, with particular attention to the details of functionality. Additionally, I employed manual testing methods to examine text summarization, analysis of text entities, and the accurate and precise storage of data. Furthermore, I ensured that the responses provided by the program corresponded to the questions, ensuring functionality accuracy and system reliability.

In conclusion, I presented innovative suggestions for the further development of the program, aiming to expand its functionality and enhance its efficiency. These include fine-tuning artificial intelligence models and introducing an automatic learning mechanism closely integrated with them.

In summary, the proposed improvements not only facilitate the expansion of the program's functionality but also significantly increase its efficiency, contributing to the system's future competitiveness and sustainable development.

9 ÁBRAJEGYZÉK

10 TÁBLAJEGYZÉK

11 IRODALOMJEGYZÉK

[1] N. Vaidya, „5 Natural Language Processing Techniques for Extracting Information,” [Online]. Available: https://blog.aureusanalytics.com/blog/5-natural-language-processing-techniques-for-extracting-information. [Hozzáférés dátuma: 2024. 12. 15.].

[2] S. IBM, „What is Natural Language Processing? | IBM,” [Online]. Available: https://www.ibm.com/topics/natural-language-processing. [Hozzáférés dátuma: 2024. 12. 15.].

[3] S. IBM, „NLP vs. NLU vs. NLG: the differences between three natural language processing concepts - IBM Blog,” [Online]. Available: https://www.ibm.com/blog/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/. [Hozzáférés dátuma: 2024. 12. 15.].

[4] M. All, „Mastering Natural Language Processing (NLP) with PyTorch: Comprehensive Guide | DataCamp,” [Online]. Available: https://www.datacamp.com/tutorial/nlp-with-pytorch-a-comprehensive-guide. [Hozzáférés dátuma: 2024. 12. 15.].

[5] A. A. Awman, „What is Named Entity Recognition (NER)? Methods, Use Cases, and Challenges | DataCamp,” [Online]. Available: https://www.datacamp.com/blog/what-is-named-entity-recognition-ner#rdl. [Hozzáférés dátuma: 2024. 12. 15.].

[6] S. Amazon, „What is Sentiment Analysis? - Sentiment Analysis Explained - AWS,” [Online]. Available: https://aws.amazon.com/what-is/sentiment-analysis/. [Hozzáférés dátuma: 2024. 12. 15.].

[7] Abid All Awman, Avinash Naviani, „Naive Bayes Classifier Tutorial: with Python Scikit-learn | DataCamp,” [Online]. Available: https://www.datacamp.com/tutorial/naive-bayes-scikit-learn. [Hozzáférés dátuma: 2024. 12. 15.].

[8] N. Barney, „What Is Sentiment Analysis (Opinion Mining)? | Definition from TechTarget,” [Online]. Available: https://www.techtarget.com/searchbusinessanalytics/definition/opinion-mining-sentiment-mining. [Hozzáférés dátuma: 2024. 12. 15.].

[9] S. Parikh, „What is Sentiment Analysis? - GeeksforGeeks,” [Online]. Available: https://www.geeksforgeeks.org/what-is-sentiment-analysis/. [Hozzáférés dátuma: 2024. 12. 15.].

[10] A. Opidi, „A Gentle Introduction to Text Summarization in Machine Learning,” [Online]. Available: https://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/. [Hozzáférés dátuma: 2024. 12. 15.].

[11] A. P. Pabitra Kumar Das, „Aspect-Based Opinion Mining,” [Online]. Available: https://devopedia.org/aspect-based-opinion-mining. [Hozzáférés dátuma: 2024. 12. 15.].

[12] M. Haller, „The Importance of Text Preprocessing in TTS,” [Online]. Available: https://beyondwords.io/blog/the-importance-of-text-preprocessing-in-tts/. [Hozzáférés dátuma: 2024. 12. 15.].

[13] S. Wikipédia, „Graféma – Wikipédia,” [Online]. Available: https://hu.wikipedia.org/wiki/Graf%C3%A9ma. [Hozzáférés dátuma: 2024. 12. 15.].

[14] S. Wikipédia, „Fonéma – Wikipédia,” [Online]. Available: https://hu.wikipedia.org/wiki/Fon%C3%A9ma. [Hozzáférés dátuma: 2024. 12. 15.].

[15] S. Ulife.ai, „Complete guide to text-to-speech with coqui TTS,” [Online]. Available: https://ulife.ai/stories/complete-guide-to-text-to-speech-with-coqui-tts. [Hozzáférés dátuma: 2024. 12. 15.].

[16] S. Amazon, „What Is Amazon Polly? - Amazon Polly,” [Online]. Available: https://docs.aws.amazon.com/polly/latest/dg/what-is.html. [Hozzáférés dátuma: 2024. 12. 15.].

[17] S. Openhab.org, „Pico Text-to-Speech - Voices | openHAB,” [Online]. Available: https://www.openhab.org/addons/voice/picotts/. [Hozzáférés dátuma: 2024. 12. 15.].

[18] pawangfg, „Explanation of BERT Model - NLP - GeeksforGeeks,” [Online]. Available: https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/. [Hozzáférés dátuma: 2024. 12. 15.].

[19] B. Muller, „BERT 101 - State Of The Art NLP Model Explained,” [Online]. Available: https://huggingface.co/blog/bert-101. [Hozzáférés dátuma: 2024. 12. 15.].

[20] N. S. N. P. J. U. L. J. A. N. G. L. K. I. P. Ashish Vaswani, „[1706.03762] Attention Is All You Need," 2017. 08. 02.. [Online]. Available: https://arxiv.org/abs/1706.03762. [Hozzáférés dátuma: 2024. 12. 15.].

[21] J. Soni, „What is Llama 2: Meta's AI explained - Dexerto," [Online]. Available: https://www.dexerto.com/tech/what-is-llama-2-2224223/. [Hozzáférés dátuma: 2024. 12. 15.].

[22] A. Sagiodev, „A brief history of LLaMA models - AGI Sphere," 11 08 2023. [Online]. Available: https://agi-sphere.com/llama-models/#LLaMA_base_model. [Hozzáférés dátuma: 2024. 12. 15.].

[23] (. meta-llama, „meta-llama (Meta Llama)," Meta, [Online]. Available: https://huggingface.co/meta-llama. [Hozzáférés dátuma: 2024. 12. 15.].

[24] T. Latterner, „LLM & GPT: what are they, and how do they work? | by Thomas Latterner | Medium," [Online]. Available: https://medium.com/@thomas.latterner/llm-gpt-what-are-they-and-how-do-they-work-2df1b5925f6. [Hozzáférés dátuma: 2024. 12. 15.].

[25] B. Lutkevich, „What is GPT-3? Everything You Need to Know - TechTarget," [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/GPT-3. [Hozzáférés dátuma: 2024. 12. 15.].

[26] s. GeeksforGeeks, „From GPT-3 to GPT-4: Evolution and Innovations in Large Language Models - GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/from-gpt-3-to-gpt-4-evolution-and-innovations-in-large-language-models/. [Hozzáférés dátuma: 2024. 12. 15.].

[27] S. Bansall, „Agents in Artificial Intelligence - GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/agents-artificial-intelligence/. [Hozzáférés dátuma: 2024. 12. 15.].

[28] S. Insidr.Ai, „AI Agents – Full Guide to Intelligent Agents for Beginners," [Online]. Available: https://www.insidr.ai/ai-agents-full-guide-to-intelligent-agents-for-beginners/. [Hozzáférés dátuma: 2024. 12. 15.].

[29] C. Hashemi-Pour, „What is an intelligent agent? | Definition from TechTarget," [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/agent-intelligent-agent. [Hozzáférés dátuma: 2024. 12. 15.].

[30] dmin@OracleCMS, „What Are the Different Types of Intelligent Agents?,” [Online]. Available: https://www.oraclecms.com/blog/different-types-of-intelligent-agents/. [Hozzáférés dátuma: 2024. 12. 15.].

[31] S. Indeed, „What Is an Agent in Artificial Intelligence? (Plus Types) | Indeed.com,” [Online]. Available: https://www.indeed.com/career-advice/career-development/agent-in-artificial-intelligence. [Hozzáférés dátuma: 2024. 12. 15.].

[32] H. Dhaduk, „6 Types of AI Agents: Exploring the Future of Intelligent Machines,” [Online]. Available: https://www.simform.com/blog/types-of-ai-agents/. [Hozzáférés dátuma: 2024. 12. 15.].

[33] D. Pena, „An Introduction to LangChain: AI-Powered Language Modeling,” [Online]. Available: https://www.sitepoint.com/an-introduction-to-langchain-ai-powered-language-modeling/. [Hozzáférés dátuma: 2024. 12. 15.].

[34] S. LangChain, „Introduction | Langchain,” [Online]. Available: https://python.langchain.com/docs/introduction/. [Hozzáférés dátuma: 2024. 12. 15.].

[35] namaldesign, „Introduction to Langchain - GeeksforGeeks,” [Online]. Available: https://www.geeksforgeeks.org/introduction-to-langchain/. [Hozzáférés dátuma: 2024. 12. 15.].

[36] M. All, „How to Build LLM Applications with LangChain | DataCamp,” [Online]. Available: https://www.datacamp.com/tutorial/how-to-build-llm-applications-with-langchain. [Hozzáférés dátuma: 2024. 12. 15.].

[37] S. LangChain, „Model I/O | Langchain,” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/model_io/. [Hozzáférés dátuma: 2024. 12. 15.].

[38] S. LangChain, „Prompts | Langchain,” [Online]. Available: https://python.langchain.com/api_reference/core/prompts.html#langchain-core-prompts. [Hozzáférés dátuma: 2024. 12. 15.].

[39] S. LangChain, „Output parsers | Langchain,” [Online]. Available: https://python.langchain.com/docs/concepts/output_parsers/. [Hozzáférés dátuma: 2024. 12. 15.].

[40] S. LangChain, „Retrievers | Langchain,” [Online]. Available: https://python.langchain.com/docs/integrations/retrievers/. [Hozzáférés dátuma: 2024. 12. 15.].

[41] S. LangChain, „Retrieval | Langchain,” [Online]. Available: https://python.langchain.com/docs/concepts/retrieval/. [Hozzáférés dátuma: 2024. 12. 15.].

[42] S. LangChain, „Chains | Langchain,” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/chains/. [Hozzáférés dátuma: 2024. 12. 15.].

[43] S. LangChain, „Memory | Langchain,” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/memory/. [Hozzáférés dátuma: 2024. 12. 15.].

[44] S. LangChain, „Agents | Langchain,” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/agents/. [Hozzáférés dátuma: 2024. 12. 15.].

[45] S. Docker, „Docker overview | Docker Docs,” [Online]. Available: https://docs.docker.com/get-started/overview/. [Hozzáférés dátuma: 2024. 12. 15.].

[46] Azmat, „Docker: Components (Client, Host, Daemon, etc.) - Knoldus Blogs Docker,” [Online]. Available: https://blog.knoldus.com/docker-components/. [Hozzáférés dátuma: 2024. 12. 15.].

[47] S. Docker, „Docker daemon configuration overview | Docker Docs,” [Online]. Available: https://docs.docker.com/config/daemon/. [Hozzáférés dátuma: 2024. 12. 15.].

[48] S. Docker, „Dockerfile reference | Docker Docs,” [Online]. Available: https://docs.docker.com/engine/reference/builder/. [Hozzáférés dátuma: 2024. 12. 15.].

[49] S. Docker, „What is a Container? | Docker,” [Online]. Available: https://www.docker.com/resources/what-container/. [Hozzáférés dátuma: 2024. 12. 15.].

[50] S. Sysdig, „What is Docker architecture? – Sysdig,” [Online]. Available: https://sysdig.com/learn-cloud-native/container-security/what-is-docker-architecture/. [Hozzáférés dátuma: 2024. 12. 15.].

[51] S. Docker, „Use Docker Compose | Docker Docs,” [Online]. Available: https://docs.docker.com/get-started/08_using_compose/. [Hozzáférés dátuma: 2024. 12. 15.].

[52] S. Docker, „Docker Engine overview | Docker Docs,” [Online]. Available: https://docs.docker.com/engine/. [Hozzáférés dátuma: 2024. 12. 15.].

[53] iamrj846, „Docker Networking - GeeksforGeeks,” [Online]. Available: https://www.geeksforgeeks.org/basics-of-docker-networking/. [Hozzáférés dátuma: 2024. 12. 15.].

[54] S. Docker, „Volumes | Docker Docs,” [Online]. Available: https://docs.docker.com/storage/volumes/. [Hozzáférés dátuma: 2024. 12. 15.].

[55] Z. Hira, „Kubernetes VS Docker Swarm – What is the Difference?,” [Online]. Available: https://www.freecodecamp.org/news/kubernetes-vs-docker-swarm-what-is-the-difference/. [Hozzáférés dátuma: 2024. 12. 15.].

[56] S. Microsoft, „Setting up Visual Studio Code,” [Online]. Available: https://code.visualstudio.com/docs/setup/setup-overview. [Hozzáférés dátuma: 2024. 12. 15.].

[57] S. Docker, „Docker Desktop | Docker Docs,” [Online]. Available: https://docs.docker.com/desktop/. [Hozzáférés dátuma: 2024. 12. 15.].

[58] Mattwojo, „What is Windows Subsystem for Linux | Microsoft Learn,” Microsoft, [Online]. Available: https://learn.microsoft.com/en-us/windows/wsl/about. [Hozzáférés dátuma: 2024. 12. 15.].

[59] S. Svelte, „Introduction • Docs • Svelte,” Svelte, [Online]. Available: https://svelte.dev/docs/kit/introduction. [Hozzáférés dátuma: 2024. 12. 15.].

[60] meta-llama, „meta-llama/Llama-3.2-1B · Hugging Face,” [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-1B. [Hozzáférés dátuma: 2024. 12. 15.].

[61] H. Hakim, „Enhancing AI Language Models: A Conceptual Overview of LLM Chains | by Hanit Hakim | Dev Genius,” [Online]. Available: https://blog.devgenius.io/llm-chains-theoretical-overview-5f8fdad3f081. [Hozzáférés dátuma: 2024. 12. 15.].

[62] H. K. Ilter, „What is Tensor? - ILTER,” [Online]. Available: https://hkilter.com/index.php?title=What_is_Tensor%3F. [Hozzáférés dátuma: 2024. 12. 15.].

[63] D. Jones, „Containers vs. Virtual Machines (VMs): What's the Difference? | NetApp Blog,” [Online]. Available: https://www.netapp.com/blog/containers-vs-vms/. [Hozzáférés dátuma: 2024. 12. 15.].