Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Jeges Gábor

NAGY NYELVI MODELL ALAPÚ CHATBOT SPECIÁLIS FELADATRA

KONZULENS

Dr. Forstner Bertalan

BUDAPEST, 2024

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Jeges Gábor**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2024. 01. 04.

...…….
Jeges Gábor

Összefoglaló

Egy tárgyfelelőshöz rengeteg kérdés érkezik, főleg, ha az ember a diploma megszerzéséhez szükséges szakmai gyakorlatért felelős. Ehhez még egy alaptárgy esetében is számtalan szabály tartozik, mind törvényi, mind egyetemi előírásként. A tárgyfelelős a hallgatóktól érkezett kérdésekre csak némi késéssel, esetleges időben változó információk megismerése után tud válaszolni. Ennek kiváltására alkalmas lehet egy chatbot, mely tud válaszolni a szakmai gyakorlattal kapcsolatos kérdésekre.

Az elmúlt időszakban hatalmas fejlődésnek indultak a nagy nyelvi modellek. Népszerű kutatási területként rengeteg megoldási lehetőség áll rendelkezésre egy speciális feladatra képes chatbot elkészítéséhez. A paraméterek hatékony újraszámításával finomhangolható egy modell, annak teljes újratanítása nélkül, vektoradatbázisok segítségével pedig külső forrásokhoz lehet csatlakoztatni, így tanítási idő utáni vagy saját adatokkal is tud dolgozni. Ezen a gyorsan fejlődő területen bármelyik nap megjelenhet valami új, ami más megvilágításba helyezi az eddigi technikákat.

A dolgozat célja egy olyan chatbot megtervezése, mely képes válaszolni a BME Villamosmérnöki és Informatikai Karának szakmai gyakorlatával kapcsolatban felmerülő kérdésekre. Cél, hogy a chatbottal Microsoft Teams-en keresztül lehessen kommunikálni. A tervezési folyamat mellett a modellek működési elve is bemutatásra kerül.

Abstract

The person in charge of a course gets a lot of questions from students, especially when they’re responsible for the traineeship needed to obtain the degree. There are lots of rules for doing the traineeship, both from the law and the university. The course coordinator may be able to respond only after some delay, following the acquisition of information that might vary with time. A possible solution to this could be a chatbot capable of answering these.

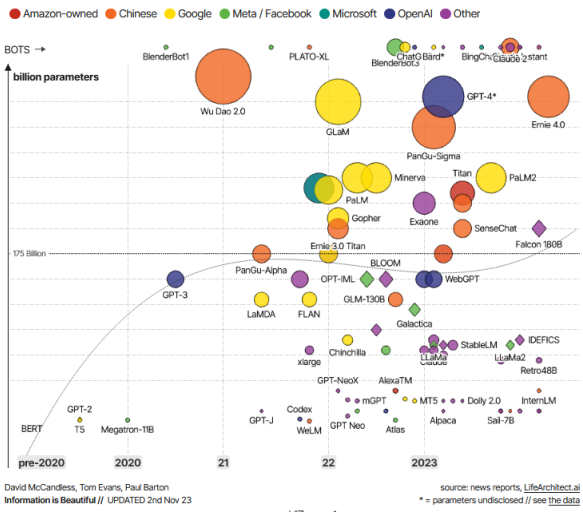
The recent period has witnessed significant advancements in large language models. There are various solutions available in this research area to create a specialized chatbot for a specific task. The model can be fine-tuned using Parameter-Efficient Fine-Tuning, without the need for complete retraining. They can also be linked to external sources using Retrieval Augmented Generation, allowing it to answer taking into account post-training or personal data. In this rapidly evolving field, new developments may appear at any time, altering existing techniques.

The thesis aims to design a chatbot capable of responding to queries related to the traineeship at the BME Faculty of Electrical Engineering and Informatics. The communication with the chatbot is set to be achieved through Microsoft Teams. Alongside the design process, the operational principles of the models will also be presented.

1 Bevezetés

2023 végét írunk. Szinte 1 éve sincs, hogy megjelent a ChatGPT és teljesen felforgatta a világot. Habár nem ez volt az első nagy nyelvi modellt használó chatbot, ez volt az első, ami a hétköznapi emberek számára elérhetővé tette azt. Alig 5 nap alatt elérte az 1 millió felhasználót, mára pedig már több, mint 150 millió embert tudott meggyőzni, hogy regisztráljanak az oldalra [1].

A megjelenést követően elég hamar a média is elkezdett foglalkozni vele. Sorra jelentek meg a „különféle foglalkozásúak félhetnek a ChatGPT-től” típusú cikkek, majd aztán ennek az ellenkezőjéről szólóak is. A nagy techcégek is természetesen még jobban ráálltak erre a területre. A Microsoft az OpenAI-ba, a ChatGPT-t is kiadó mesterséges intelligenciával foglalkozó amerikai vállalatba fektetett be. Annak szolgáltatásait az Azure-ba, a Microsoft felhőalapú platformjába is beépítették, a Binget pedig a GPT-4-gyel tették vonzóvá. A Google is megjelentette a Bard-ot, a saját chatbotját, amely a cég saját modelljére, a PaLM 2-re épül. A Meta pedig a nyáron egy félig-meddig nyílt modellel, a Llama-2-vel rukkolt elő. Ezeken kívül persze kisebb cégek is elkezdtek különböző ötleteket kidolgozni, hogy versenyelőnyt érjenek el a nagy nyelvi modelleket (angol rövidítés szerint LLM-eket) használó programokban.



1.1. ábra LLM-ek megjelenése és paraméterszáma[1]

[1] Forrás: Information is beautiful: https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/ (2023.11.15.)

Természetesen chatbotok e modellek előtt is léteztek, azonban ezek szavakra, szókapcsolatokra tudtak reagálni, esetleg előre meghatározott kérdéseket tettek fel, és azokra tudtak a felhasználók vagy megadott opciókból választani vagy egyszerűbb szöveges választ is adni. Az LLM-ek felemelkedésével (1.1 ábra) azonban lényegesen jobb és komplexebb chatbotok fejlesztése vált lehetővé.

A mai világban rengeteg helyen van olyan helyzet, ahol több emberben is felmerülnek ugyanazok vagy nagyon hasonló kérdések, ezekkel pedig megkeresnek egy embert, aki tud rá válaszolni. Ilyen lehet például egy ügyfélszolgálatos, egy banki dolgozó, egy hivatalnok, de egy tárgyfelelős is ilyen pozícióban találhatja magát.

A Villamosmérnöki és Informatikai Karon a BSc diploma megszerzéséhez szükséges a képzés során egy, szaktól függően, 6 vagy 8 hetes szakmai gyakorlaton is részt venni. Ennek felelőse a dolgozat írásának idején Dr. Blázovics László, akihez még egy átlagos tantárgyfelelősnél is több kérdés érkezik. Habár van tanszéki portál, ahol rengeteg információ megtalálható, azok nem feltétlen térnek ki mindenre vagy nem teljesen egyértelműek, hiszen nem csak egyetemi, kari, hanem törvényi szabályozás is van a gyakorlatokra vonatkozóan. Ezek miatt a hallgatók természetesen sokszor hozzá fordulnak, de sokszor hossszú időbe telik válaszolni egyéb teendői miatt, vagy mert változó információknak kell utánanéznie. Mindkét fél számára hasznos lehet tehát egy chatbot, amely képes a VIK szakmai gyakorlatával kapcsolatos kérdésekre válaszolni, hogy már csak a nagyon specifikus kérésekkel forduljanak a felelőshöz.

Korábban foglalkoztam már az egyetemen az OpenAI-jal, majd a szakmai gyakorlatomon is nagy nyelvi modellek segítségével próbáltunk egy meglévő szoftverhez egy kezdetleges AI asszisztenst készíteni, így mikor megláttam ezt a témát a tanszéki portálon, egyből felkeltette az érdeklődésemet. Úgy gondoltam jó lenne a tudásomat bővíteni ezeknek a modelleknek az alaposabb megismerésével, és emellé egy chatbotot is tervezni, ahol ezeket a gyakorlatban is ki tudom próbálni. Véleményem szerint ez a tudás az elkövetkezendő években a munkaerőpiacon is hasznosulni tud majd, ha figyelembe vesszük, milyen irányba halad a világ.

1.1 A feladat értelmezése

A dolgozat célja, hogy bemutassa a nagy nyelvi modellek elméleti hátterét, majd ezeket figyelembe véve egy chatbot tervezése és fejlesztése, mellyel Microsoft Teamsen keresztül lehet kommunikálni.

Bemutatásra kerül a nagy nyelvi modellek alapjául szolgáló transzformer architektúra felépítése, működése és fajtái, valamint röviden az, hogy miben változtatta meg az addigi természetes nyelvfeldolgozást a megjelenése. A fajtáknál azok összehasonlítására, illetve példák említésére is kitérek.

A modellek finomhangolási módszereinél a zero-, one-, és few-shot inference (amikor a modell bemenetébe 0, 1, vagy több példát írunk az elvárt kimenethez való következtetésre), teljes finomhangolás, illetve a Parameter-Efficient Fine-Tuning (röviden PEFT, mikor a paraméterek csak egy bizonyos részét változtatjuk) kerül terítékre. Tanítási idő utáni vagy saját adatokhoz pedig Retrieval Augmented Generation (RAG, valamilyen információ kinyeréssel megtámogatott szöveggenerálás) használható, ezt is ismertetem.

A különböző módszereket összehasonlítom, kifejtem, hogy melyiket és miért láttam jobbnak az adott feladathoz, vagy éppen milyen problémákba ütköztem azok tanulmányozása, tesztelése közben. A választott módszer segítségével pedig megtervezem és megvalósítom a chatbot prototípusát, amivel egy Microsoft Teams-es chaten keresztül lehet kommunikálni.

1.2 A dolgozat felépítése

A dolgozat első részében az LLM-eket mutatom be architektúrájuk, finomhangolásuk, és külső forrásokhoz kapcsolásuk sorrendben. A következő szekcióban a felhasznált technológiákról és könyvtárakról adok áttekintést. A 4. és 5. fejezetben a chatbot megtervezése és megvalósítása kerül leírásra. Végezetül összefoglalom a munkámat és a továbbfejlesztési lehetőségeket ismertetem.

2 Nagy nyelvi modellek

A nagy nyelvi modellek olyan neurális háló alapú matematikai modellek, melyek természetes nyelvű bemenetből szöveget generálnak. A kapott inputból valószínűség alapján határozzák meg a választ. Rengeteg általános nyelvi feladatra alkalmasak, köszönhetően annak, hogy hatalmas adathalmazokon vannak tanítva. Ezek olyan neurális hálók, amelyeknek akár több milliárd paramétere is lehet, amelyek értékét a hosszas és a nagy mennyiségű adattal való tanítás határozza meg [2].

2.1 Korábbi megoldások

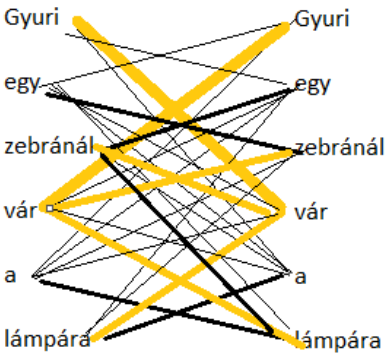
Szöveg generálására a transzformer architektúrájú modellek előtt visszacsatolásos neurális hálózatokat (recurrent neural network, röviden RNN) használtak, és használnak még ma is több helyen, például a Sirinél, a Google Fordítónál, vagy az okostelefonos gépelést segítő automatikus kiegészítéseket felajánló rendszereknél. Az RNN-ek figyelembe veszik a szavak sorrendjét a bemeneten és a korábbi információkat is, innen jön a visszacsatolás. Problémája az ilyen típusú nyelvi modelleknek azonban, hogy a nagyobb volumenű összefüggések felismerésénél már gondokba ütköznek [3]. Ennek kiküszöbölésére tökéletes megoldásként szolgál a transzformer architektúra.

2.2 Transzformer architektúra

2017-ben a Google és a Torontói Egyetem kutatói publikálták az „Attention Is All You Need” (magyarul: Csak figyelemre van szükséged) című tanulmányt [4]. A legnagyobb előnye a korábbi szöveggeneráló modelltípusokkal szemben az, hogy nem csak a szomszédos szavakkal létesített kapcsolatot veszi figyelembe, hanem hogy a bemenet minden szavát az összes többivel kapcsolati rendszerbe képezi le, mindegyikhez egy számértéket rendelve. Ezeket az értékeket az angol attention weight-nek nevezi, ez mutatja meg melyik szó, milyen szorosan kapcsolódik a másikhoz. Ezt a rendszert nevezik self-attention-nek, ami az architektúra magját adja (2.1 ábra).

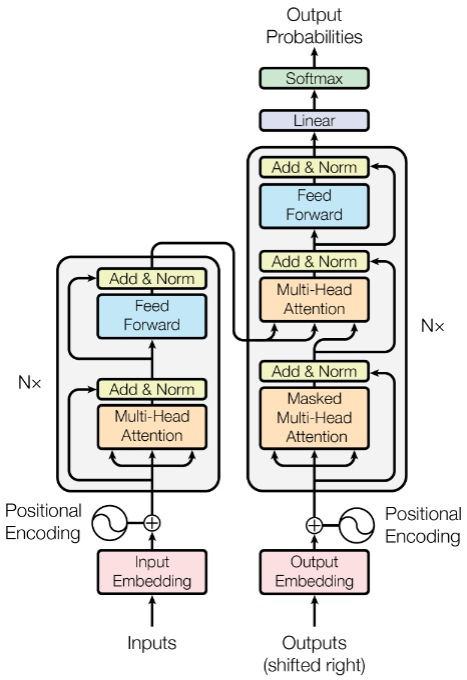
A transzformer modelleket alapvetően 2 fontos egység, az encoder (kódoló) és a decoder (dekódoló) alkotja, ahogy ez a 2.2-es ábrán is látható. A modellek természetesen nem az egész szöveget kapják meg egyben. Első lépésként tokenizálni, vagyis egységekre kell bontani a bemenetet, mindegyikhez egy token ID-t (azonosítót) rendel a tokenizáló.

Fontos, hogy tanításnál és később a modell használatakor is ugyanazt a tokenizálót használjuk.



2.1. ábra Self-attention, sárgával a vár szóhoz erősen kapcsolódó szavak

Következő lépésként az azonosítókat leképezzük egy 512 dimenziós vektorra [4]. Ez a folyamat az embedding, amit magyarra beágyazásként lehetne fordítani. Minél közelebb van egy máshoz két vektor ebben a térben, annál jobban kapcsolódnak egymáshoz. Ezekhez a vektorokhoz még hozzáadódik egy, a szavak pozícióját tároló vektor is.



2.2. ábra Transzformer architektúra[2]

[2] Forrás:Attention Is All You Need [4]

Az így kialakuló 512 dimenziós tömböket már fel tudja dolgozni az encoder és a decoder is. Mindkét egység többrétegű, melyek mindegyike összetett. Egy self-attention egységből és egy feedforward neural network-ből (röviden FNN, előrecsatolásos neurális hálózatból) áll.

A self-attention egységek multi-headed (több fejű) egységek, mindegyik fej a szöveg egy különböző aspektusára koncentrál, így alakul ki a végén a megfelelő érték. Modellenként változó, hogy mennyi, és a nyelv melyik részére koncentráló fejet tartalmaz. Az FNN egy valós számot generál ezután, amely a bemeneti adat és a neurális hálózat súlyainak kombinációját képviseli, ezt kapja meg a következő réteg. A decoder ezen kívül használja még az encoder kimenetét, szintén egy multi-headed self-attention-ben, majd a vektort adja meg az FNN-nek. A rétegek összes egysége után megtörténik a vektorok normalizációja. Az értékek a szótár összes elemére kiszámításra kerülnek, majd a dekódoló után lévő softmax réteg valószínűséget képez belőlük, a legnagyobb értékű pedig a generált token lesz [5]. A szöveg változatossága, kreativitása érdekében ezen lehet pár paraméterrel változtatni, hogy ne mindig az első szót válassza ki.

Az architektúrának vannak olyan változatai is, melyek a két fő egységből csak az egyiket tartalmazzák. A csak encoder modelleket lehet használni klasszifikációs feladatokra, kérdés megválaszolásra, egy ilyen típusú modell például a BERT (Bidirectional Encoder Representations from Transformers) [6]. A csak dekódolós modellek a legtöbbször használtak manapság, általános célra használhatóak, ebbe a kategóriába tartoznak a GPT (Generative Pre-trained Transformer) modellek is [7]. Az encoder-decoder modelleket főként természetes nyelvek közötti fordításra lehet jól használni, egy ilyen a BART [8].

A modelleket előre betanítják egy általános szintre, de közel sem optimálisan. A „Chinchilla tanulmány” [9] rámutatott, hogy az LLM-ek többsége túlparametrizált, viszont a tanító adathalmazuk meg túl kicsi ehhez képest. A kutatók megállapították, hogy körülbelül 20-szorosának kell lennie az adathalmazban lévő tokenek számának a modell paramétereihez képest.

Az architektúra előnye a korábbi RNN-hez képest a kontextusfüggő szövegértelmezésen kívül az, hogy párhuzamos feldolgozásra is képes, ezért skálázható.

A kimenet milyenségét befolyásolni tudjuk bizonyos értékek módosításával. Egyik ilyen a temperature (hőmérséklet), ami egy nemnegatív érték, és azt tudjuk vele

állítani, hogy mennyire legyen kreatív a válasz. 0-nál konzisztens válaszokat kapunk, kreativitás nélkül. A top-p és a top-k paraméterek is azt szabályozzák, melyik tokenek legyenek figyelembe véve az utolsó rétegnél, a softmaxnál. A top-p (más néven nucleus) azt mondja meg, hogy a kiválasztott tokenek összvalószínűsége mit kell meghaladjon. A top-k-val a legvalószínűbb k darab paraméterből választást tudjuk kikényszeríteni. Mindkét esetben a kisebb érték kevésbé változatos szöveget fog eredményezni. Ezeken kívül a maximálisan generálandó tokenek számát is megadhatjuk.

2.3 Prompt engineering

Prompt engineering-nek nevezik azt a folyamatot, amikor a modell bemenetét úgy alakítjuk, hogy a válasz az általunk kívánthoz közelebb álljon. Ez lehet annyi, hogy a kimenetet szeretnénk speciális formában (például CSV) megkapni, de a teljesítmény javítása érdekében is lehet a promptot (a szöveg, amit a modell bemenetként megkap) módosítani.

Az eredmények javítása érdekében használható egyik prompt engineering módszer az in context learning (kontextusban történő tanulás), ami azt jelenti, hogy a bemenetbe példákat téve a modell számára jelezzük, milyen választ várunk egy-egy promptra.

A nagyobb modellek jó eredményeket adhatnak csak az utasítások alapján is, példák nélkül (zero-shot inference), de használható itt is one-shot vagy a few-shot inference (egy vagy több példás következtetés). Kisebb modelleknél pedig mindenképp érdemes utóbbiakat megpróbálni. Egy ilyen eset lehet, ha szöveget szeretnénk összegezni. Ekkor a prompt elején megadunk több rövidebb szöveget, és az összefoglalást mindegyik után.

2.4 Finomhangolási módszerek

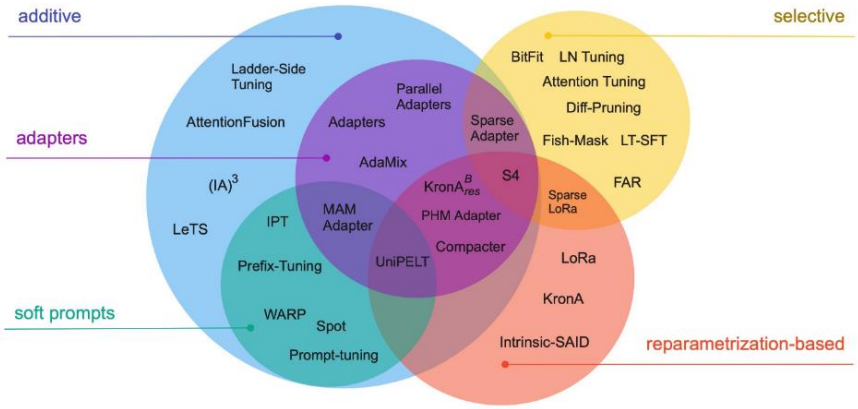
Ha az előző szekcióban taglalt megoldás nem vezetne kellően jó eredményekre, a modell továbbtanítását is választhatjuk. Ezen kívül a nagy nyelvi modellek egyik hátránya, hogy önmagukban nem férnek hozzá a tanítási idő utáni, illetve a saját privát dokumentumainkhoz. Ezek használatára is egy megoldás a modell finomhangolása.

2.4.1 Teljes finomhangolás

A modellek teljes finomhangolása nagyon erőforrásigényes feladat mind idő, mind számítási költség szempontjából. Prompt-completion párokat (bemenet-kimenet párokat) képezünk, majd tokenizáljuk őket. Ezekkel tudjuk aztán a modellt futtatva az adathalmazon frissíteni a súlyokat. A finomhangolt modelleket szokták Instruct LLM-eknek is nevezni. Finomhangolás esetén figyelni kell a „catastrophic forgetting" jelenségre (katasztrofális felejtésre). Ez azt jelenti, hogy a modell túlságosan rátanul az új feladatra, és emiatt romlik az általános teljesítménye is. Ennek kiküszöbölésére egyszerre több feladatra is taníthatjuk a botot, amihez viszont még több adatra van szükség [10].

2.4.2 PEFT

A Parameter-Efficient Fine-Tuning a paraméterek csak egy kis részére koncentrál vagy a meglévő modellhez ad hozzá újabb rétegeket, de a paraméterek egy része mindig befagyasztásra kerülnek [11]. Több fajtája is van (2.3 ábra):



2.3. ábra A PEFT típusai, forrás: [11]

- Szelektív: a meglévő rétegek egy részét finomhangolja
- Additív
    - Adapter: a self-attention vagy az FFN után egy adaptert rak
    - Soft prompt: a modellt érintetlenül hagyja, csak a szövegbemenetet változtatja meg (pl. virtuális tokenekkel)

- Újraparaméterezés: alacsony szintű reprezentációk használatával újraparametrizálja a modellt, egy ilyen típusú PEFT módszer a Low-Rank Adaptation of Large Language Models (LoRA), amely a súlyfrissítéseket két mátrix szorzatával végzi el, amit az eredeti súlyokhoz ad hozzá. Ezeket a mátrixokat alacsony rangú dekompozícióval hozza létre [12].

2.5 Külső források használata

A finomhangolás mellett a másik megoldás tanítási idő utáni vagy saját adatok használatára a Retrieval Augmented Generation (RAG, információvisszanyeréssel támogatott szöveggenerálás). RAG használatakor a modellhez nem nyúlunk, csak valamilyen módon általa nem ismert adatokhoz adunk neki hozzáférést [13].

2.5.1 Vektoradatbázisok

Saját dokumentumok használatához a legjobb módszer, ha a modell elé teszünk egy vektoradatbázist, amiben a szükséges adatainkat tároljuk. Az adatbázisban a dokumentumokat részekre bontva, embedding segítségével leképezve tároljuk. A felhasználótól kapott kérdést ugyanazzal a módszerrel, mint az adatokat, vektorrá alakítjuk, és a legkisebb távolságra lévőben megtalálhatjuk a releváns információkat.

Természetesen nem mindig a legjobb értékű lesz a legrelevánsabb egy-egy adott kérdésnél, ezért több metódus is van adataink kinyerésére az adatbázisból. A legegyszerűbb megoldás, hogy ne csak az első, hanem több forrást adunk vissza. Ennek hátránya, hogy sok esetben nem derül ki új információ, mert ezek a források is nagyon hasonlítanak egymásra. Ezt a problémát orvosolja a maximal marginal relevance (MMR), amely úgy működik, hogy a megadott számú legrelevánsabb dokumentumrészletből választ ki minél különbözőbbeket, például 8-ból az 5 legeltérőbbet. A vektoradatbázisból így kinyert szövegeket a promptba ágyazva az LLM meg tudja válaszolni a kapott kérdést.

2.5.2 Programok és web elérése

A saját adatok használatához hasonlóan itt is a prompt manipulációjával érjük el azt, amit akarunk. Sokszor több hívást is intézünk ilyenkor a modell felé. Először is, hogy eldöntsük, kell-e külső eszközökhöz nyúlnunk. Ha igen, akkor meg kell fogalmaznia az LLM-nek, hogy hogyan. Ez lehet egy SQL lekérdezés formájában, vagy egy API hívás

saját alkalmazásunk vagy az internet felé is. A megkapott adatokat ugyanúgy a következő promptba tudjuk ágyazni, mint az előző esetben, így megválaszolásra kerül a kérdés.

Ilyen módszerrel nem csak adatok lekérdezésére, de akár kérések teljesítésére is alkalmassá tehetjük a modellünket, például asztal- vagy szobafoglalásra, ételrendelésre. Fejlettebb matematikai programokhoz is hozzá tudjuk kötni, hogy tényleges, komplex műveleteket is végrehajthasson programunk.

3 Felhasznált technológiák, könyvtárak

3.1 Python

A Python egy magas-szintű, általános célú programozási nyelv. 3 fő verziója van, melyek nem kompatibilisek egymással. A Pythont dinamikus típusú nyelv, valamint a memóriakezelést is megoldja szemétgyűjtéssel. Alkalmas egyaránt objektum-orientált, illetve funkcionális programozásra is. Könnyen olvasható, nagy közösségi támogatás övezi, rengeteg könyvtárat készítenek hozzá, melyeket a pip csomagkezelővel egyszerűen el tud érni bárki. Ezek miatt sok, nem informatikával foglalkozó ember is megtanulja és használja is a munkájához.[3]

3.1.1 Beautiful Soup

Ez egy olyan Python könyvtár[4], ami webes tartalmak (HTML és XML dokumentumok) parse-olására (értelmezésére, feldarabolására) alkalmas. Használhatjuk bizonyos HTML tagek kinyerésére, például az oldalon lévő címekre van szükségünk, akkor a <h1> - <h6> címkéket szedjük ki, vagy használhatjuk akár egy megadott class attribútummal rendelkező részekhez is.

3.1.2 Pandas

Ez az egyik legnépszerűbb adatelemzés céljából készített keretrendszer Pythonon belül[5]. Ez a nyílt forráskódú az adatok strukturálását és módosítását teszi nagyon egyszerűen lehetővé. Ennek alapegysége a DataFrame típus, ami kétdimenziós szerkezetben, táblázatszerűen tárolja az adatokat. Többféle formátumú adatok beolvasására és mentésére is lehetőséget nyújt: szövegfájlok, CSV, Excel és SQL adatbázisok. A dataframe-kkel tudunk adatbázistáblaszerűen műveleteket végezni: összefésülni adathalmazokat, valamelyik mező értéke szerint csoportosítani, szűrni.

---

[3] Python honlap: https://www.python.org/

[4] https://pypi.org/project/beautifulsoup4/

[5] https://pandas.pydata.org/

3.1.3 Matplotlib

Ezt a Python könyvtárat adatok vizualizációjára használják.[6] Számos diagramtípust létre lehet vele hozni, statikusakat, animáltakat, sőt akár 3 dimenziósakat is. Az ábrákat rengeteg szempont alapján testre lehet szabni. Természetesen ez is nyílt forráskódú projekt. A Pandas-ba integtrálva van, így könnyen lehet használni dataframekkel is.

3.2 Jupyter notebook

A Project Jupyter egy nyílt forráskódú projekt, aminek célja, hogy támogassa az interaktív tudományos számításokat rengeteg programnyelven[7]. Főként Pythonnal használják, 2014-ben az IPython-ból született meg. Segítségével notebookokat hozhatunk létre, mely input és output mezőket tartalmaz. A celláknak lehetnek közös változóik, de hatalmas előny egy normál Python fájlhoz képest, hogy minden mező külön futtatható, így sokkal gyorsabbá tehetünk folyamatokat, egy egész szkriptfájl futtatását mellőzve.

Lehetőség van vizuális kimenetekre is, például grafikonokra, vagy egy gradio interfész is tud futni egy output cellában. A fájlkiterjesztés ipynb, a fájl valójában JSON objektumokat tartalmaz, mindegyik egy-egy cellát reprezentál, meghatározza a típusát, a benne lévő kódot vagy szöveget, a kimenetet és azt is, hányadiknak volt lefuttatva utoljára. Úgy működik, hogy egy böngésző alapú read–eval–print loop (REPL) kernel megy a háttérben, kódunk ezen fut le.

3.3 CUDA

A CUDA (Compute Unified Device Architecture) egy NVIDIA által fejlesztett számítási platform és programozási modell. Lehetővé teszi a videokártyákat nem csak grafikus feladatok, hanem általános számítási műveletek végrehajtására is. A CUDA-nak köszönhetően a GPU (videokártya) párhuzamos feldolgozási képességeit ki tudják használni programok. Emiattt is alkalmazzák előszeretettel a gépi tanulásnál, hiszen nagyon fel tudja gyorsítani a futtatást. A TensorFlow és a PyTorch könyvtárak is

[6] https://matplotlib.org

[7] https://jupyter.org/about

támogatják, így Pythonból is könnyen elérhetőek az előnyei. Kizárólag NVIDIA GPU-kkal kompatibilis.[8]

3.4 LangChain

A LangChain[9] egy keretrendszer, amelyet a nagy nyelvi modelleket használó alkalmazások létrehozásának egyszerűsítésére használnak. Az LLM-ek magukban csak a kapott promptra tudnak válaszolni, viszont több dologgal összekapcsolva megsokszorozható a hasznosságuk. A LangChain célja ezeknek a kihívásoknak a megoldása azáltal, hogy olyan könyvtárt biztosít, amely lehetővé teszi egy nyelvi modell számára, hogy kölcsönhatásba lépjen a környezetével, például parancsokat hajtson végre, kéréseket küldjön, adatokat frissítsen.

Nyílt forráskódú projekt[10], elérhető Pythonban és JavaScriptben is, számos más eszközt is integrál és használható rajta keresztül, egy egységes interfész alatt, ilyenek például az OpenAI API és a Hugging Face a modelleknél vagy a különböző felhőszolgáltatások.

A LangChain a következő modulokból épül fel:

- **Model I/O**: Ez a modul lehetővé teszi a LangChain számára, hogy interakcióba lépjen bármilyen nyelvi modellel, lehet ez akár OpenAI-os GPT vagy Hugging Face-s is, ahogy az előbb említettem. A modullal bemeneti sablonok kezelése, létrehozása, illetve a kimenet parse-olása (megfelelő sémába alakítása) is megoldható.
- **Retrieval** (Adatlekérdezés): A modul segítségével az adatokat átalakíthatjuk, feldarabolhatjuk kisebb részletekre, adatbázisokban tárolhatjuk és azokból lekérdezésekkel hozhatjuk vissza azokat. Az adatokat rengeteg formában (akár PDF-ként, szöveges fájlként, CSV-ként stb.), betölthetjük. Forrásaik lehetnek lokálisak, de az internetről is származhatnak, még a saját felhőben lévő

[8] CUDA: https://developer.nvidia.com/cuda-zone

[9] LangChain honlap: https://www.langchain.com/

[10] https://github.com/langchain-ai

tárhelyünkből is használhatjuk a dokumentumokat. Többféle adatbázistípust is támogat.

- **Agents** (Ügynökök): Az ügynök modul segítségével a nyelvi modellek eldönthetik, melyik lépéseket vagy akciókat kell végrehajtaniuk a problémák megoldásához. Ezt úgy éri el a LangChain, hogy megfogalmazzuk a lehetséges eszközök leírását, hogy mikre képesek, majd a bemenetet és az eszközöket egy promptba téve az LLM eldönti, mit kell tenni.
- **Memory** (Memória): Alapesetben egy nyelvi modell csak az aktuális üzenetet kapja meg, ezért kell valami, ami tárolja a beszélgetés előzményeit. Ez a modul ehhez nyújt segítséget. Többféle memória is van: az utolsó k darab kérdés-válasz párt tároló, x tokennyi szöveget eltároló, és van, ami egy modell segítségével összefoglalja a korábbi beszélgetést, és azt tárolja el. Az előzményeket a memóriából kinyerve meg lehet adni a bemenetbe, amit az LLM figyelembe tud venni.
- **Chains** Láncok: Ezzel a modullal lehet összekapcsolni az összes többi elemet egy egész lánccá, így létrehozva a programunkat. Vannak különböző célokra létrehozottak is már, de magunk is építhetünk sajátot. vagy módosíthatjuk a meglévők különböző paramétereit (például beépített prompt).

A LangChaint fogom használni a chatbot különböző részeinek: memória, vektoradatbázis és nagy nyelvi modell összeköttetésére, valamint a prompt sablonok kezelésére.

3.5 Hugging Face

A Hugging Face egy francia-amerikai vállalat, melynek célja elérhetővé tenni mindenki számára a gépi tanulást. Az általuk üzemeltett platform[11] lehetővé teszi nyílt forráskódú modellek, adathalmazok megosztását. A modellek finomhangolását is támogatja különböző könyvtárakkal, valamint azok kiértékelését is. Nem csak nyelv-, hanem képfeldolgozással is foglalkoznak.

[11] https://huggingface.co/

A platformon számos cikk megtalálható a mesterséges intelligenciával kapcsolatban, kurzusokat is lehet végezni. Magyarázóvideók, illetve a könyvtáraikhoz saját útmutatók is elérhetőek. Ezeket a könyvtárakat Python és JavaScript környezetből is tudjuk használni, majd a „Spaces” szekción belül akár közzé is tehetjük saját demóinkat. Alább megemlítem az általam is használt részeit a platformnak.

3.5.1 Transformers

Ennek a könyvtárnak a használatával tudjuk betölteni bármelyik Hugging Face-n publikált modellt. Egységes interfészt biztosít, az AutoModel és AutoTokenizer osztályok egyszerű kezelést tesznek elérhetővé számunkra, így könnyen megoldhatunk különböző feladatok, mint például szöveggenerálás, fordítás vagy klasszifikáció. A PyTorch és a TensorFlow is támogatva van. A Google, a Microsoft, az Amazon és a Meta is mind publikálják itt több modelljüket is, amit esetleges hozzáférés-igénylés és egy Hugging Face-es API kulcs generálás után már használni is tudunk.

3.5.2 PEFT

A Hugging Face támogatja a modellek finomhangolását is, ennek a feladatnak a megkönnyítésére hozták létre. Nevéből is adódóan a PEFT technikák alkalmazására használható.

3.5.3 Datasets

A publikált adathalmazok betöltésére, újak megosztására használható. Adataink lehetnek szöveges, hangi vagy képi formában is akár.

3.5.4 Gradio

A Hugging Face rájött, hogy igény lenne az ő eszközeikkel készített programokhoz gyors, hatékony grafikus interfészeket kapcsolni, így elkészítették ezt a könyvtárat, mellyel akár 2 sor kódból is egy teljes chat interfészt létre tudunk hozni egy gyors prototípushoz.

3.6 OpenAI

2015-ben alapított amerikai mesterséges intelligenciával foglalkozó kutatószervezet, amely a nonprofit OpenAI Inc.-ből és a profitorientált OpenAI LP leányvállalatából áll. Céljuk, hogy a jövőben az artificial general intelligence (AGI,

általános mesterséges intelligencia: elméleti intelligencia, ami bármilyen emberi intellektuális feladatot meg tud oldani) biztonságos legyen, és az emberiség javát szolgálja[12].

2020-ban jelentették be a GPT-3-at, de igazán csak 2022 év végén váltak ismertté, mikor a ChatGPT is elérhetővé vált a GPT-3.5-ös modelljükkel. 2023-ban megjelent a GPT-4 is, mely már vizuális inputot is tudott fogadni. Ezeken kívül vannak finomhangolható modelljeik, van a DALL-E 2, mely szövegből képes képek generálására. A 2023 októberében jelent meg a DALL-E 3, amit már közvetelenül a ChatGPT-ből érhetünk el, de csak az előfizetéssel rendelkező ChatGPT Plus felhasználók. A Whisper egy nyílt forráskódú, többnyelvű automatic speech recognition (ASR, automatikus beszédfelismerő), más néven speech-to-text (beszédből szöveg) modelljük, amelyet akár lokálisan is használhatunk [14].

Modelljeiket az OpenAI API-n keresztül tudjuk használni, amely elérhető több keretrendszerből is. Hivatalosan csak Node.js és Python könyvtáruk van, valamint a Microsoft cégbe vetett befektetései nyomán az Azure-ön keresztül .NET-ből, JavaScriptből, Javaból és Goból is használhatóak. A közösségnek köszönhetően azonban más környezetekből (például Kotlinból és Swiftből mobilon, játékok fejlesztésénél Unity-ből és Unreal Engine-ből is stb.) is egyszerűen elérhető[13].

3.7 Chroma

A Chroma egy nyílt forráskódú beágyazott (embedding) vektorok tárolására alkalmas adatbázisfajta[14]. Kifejezetten nagy nyelvi modelleket használó alkalmazásokhoz fejlesztették ki. Elérhető JavaScriptből, TypeScriptből és Pythonból. Az adatbázisban együtt tárolja a vektorokat a dokumentumokkal. Képes multimodális gyűjtemények létrehozására is, tehát szöveg mellett például képeket is használhatunk. Embedding függvényként megadhatjuk az OpenAI-ét, Google PaLM-ét, bármilyen Hugging Face-est vagy Instructort[15] is. A Chroma vektoradatbázis integrálva van

[12] Az OpenAI oldaláról, https://openai.com/about

[13] https://platform.openai.com/docs/libraries

[14] https://docs.trychroma.com/

[15] Egy nyílt forráskódú módszer: https://github.com/xlang-ai/instructor-embedding

LangChain-ben is, így könnyen használható azzal a könyvtárral együtt. A Chroma-t fogom a különböző adatforrások tárolására alkalmazni.

3.8 Google Colab

A Google Colab egy, a Google által üzemeltett Jupyter Notebook szolgáltatás.[16] Ingyenesen használható, de az erősebb szervergépekért már elő kell fizetni. Akár nagyon erős, gépi tanulásra specializált A100 és V100 Tensor Core GPU-kat vagy Tensor Processing Unitokat (TPU-kat, egy Google által gépi tanulásra fejlesztett hardver) is használhatunk. Google Drive is csatlakoztatható, így nem kell minden alkalommal forrásfájljainkat feltöltenünk az újonnan kapott virtuális gépre. Mivel pár modellhez és finomhangoláshoz nem volt elegendő a laptopom memóriakapacitása (16 GB) és a videokártya-teljesítménye se, ezért a szakdolgozatkészítés második felére előfizettem a Google Colab Pro-ra, és az erőforrásigényesebb feladatokat ott oldottam meg.

3.9 Azure

Az Azure a Microsoft felhőszolgáltatása[17]. Rengeteg dolgot megvalósíthatunk a segítségével. Telepíthetünk rá Windows Server-t, SQL Servert. Web appokat, chatbotokat üzemeltethetünk ezen keresztül stb. Rengeteg mesterséges intelligencia-szolgáltatást is elérhetünk ezen keresztül, köztük az OpenAI-t vagy a Hugging Face-t is. Az alkalmazások könnyen skálázhatóak. Ingyenesen kipróbálható, de mindennapi használat esetén már fizetni kell a programok futtatásáért. Egy Azure Botot sokféle felülethez hozzá tudunk kötni: Teams, Facebook, Slack, Telegram, egy email címhez is akár.

3.10 Microsoft Bot Framework

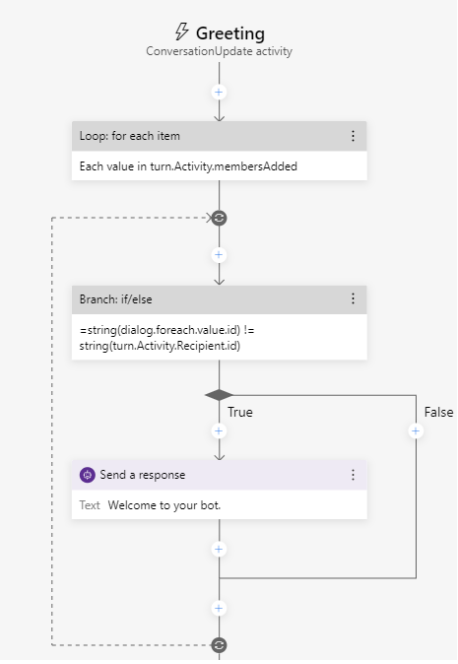
A Microsoft Bot Framework a Microsoft által, botok készítésére kifejlesztett keretrendszer. Segítségével Azure Bot Service-be telepíthető botokat készíthetünk. Elérhető JavaScript, C# és Python nyelven. Rengeteg sablon kód elérhető hozzá. A botoknak az ActivityHandler osztályból kell leszármazniuk (ez lehet specifikusan például

[16] https://colab.google/

[17] https://portal.azure.com

TeamsActivityHandler is), és különböző eseménykezelő függvényeket kell felüldefiniálniuk, hogy tudjanak reagálni a felhasználó inputjaira.

A chatbotokat lokálisan is lehet tesztelni a Bot Framework Emulator nevű programmal, amely a localhoston elindított botra tud kapcsolódni, és így biztosít egy chat felületet üzenetek küldésére fogadására, valamint a hálózati kommunikáció vizsgálatára. A Bot Framework Composerrel akár grafikus felületen is készíthetünk C# és JavaScript botokat folyamatábrák segítségével (3.1 ábra).[18]



3.1. ábra Üdvözlés példa a Composerben

Azure-ben a bot használatához egy Resource Groupon (erőforráscsoporton) belül létre kell hozni egy Azure Botot, és egy App Service-t. A kódot az utóbbiba kell telepíteni, az Azure Botot pedig a megfelelő API endpointra (ez alapesetben a /api/messages) kell kötni, illetve a csatornákat beállítani, ahonnan elérhetővé akarjuk tenni.

[18] https://dev.botframework.com/

4 A chatbot megtervezése

Az már a feladat megfogalmazásának pillanatában egyértelmű volt, hogy programnyelvnek a Pythont fogom használni, hiszen a mesterséges intelligenciás keretrendszereket nagy részét ehhez a nyelvhez készítik el a könnyű olvashatósága, egyszerűsége és közösségi támogatottsága miatt. A Python 3.11-es verziót választottam, mert ez volt az 1 éve kint lévő stabil verzió a munka kezdetekor, így minden szükséges könyvtár elérhetőnek gondoltam benne a félév kezdetekor.

4.1 Az elmélet gyakorlati megismerése

Az első feladatként azt tűztem ki magamnak, hogy az elméleti anyagot a gyakorlatba is átvigyem. Ehhez első lépésként egy generatív mesterséges intelligenciáról (MI-ről) szóló DeepLearning.AI (Andrew Ng[19] által alapított MI oktatással foglalkozó amerikai vállalat) kurzust[20] néztem meg, és az ahhoz kapcsolódó feladatokat oldottam meg, kísérleteztem velük.

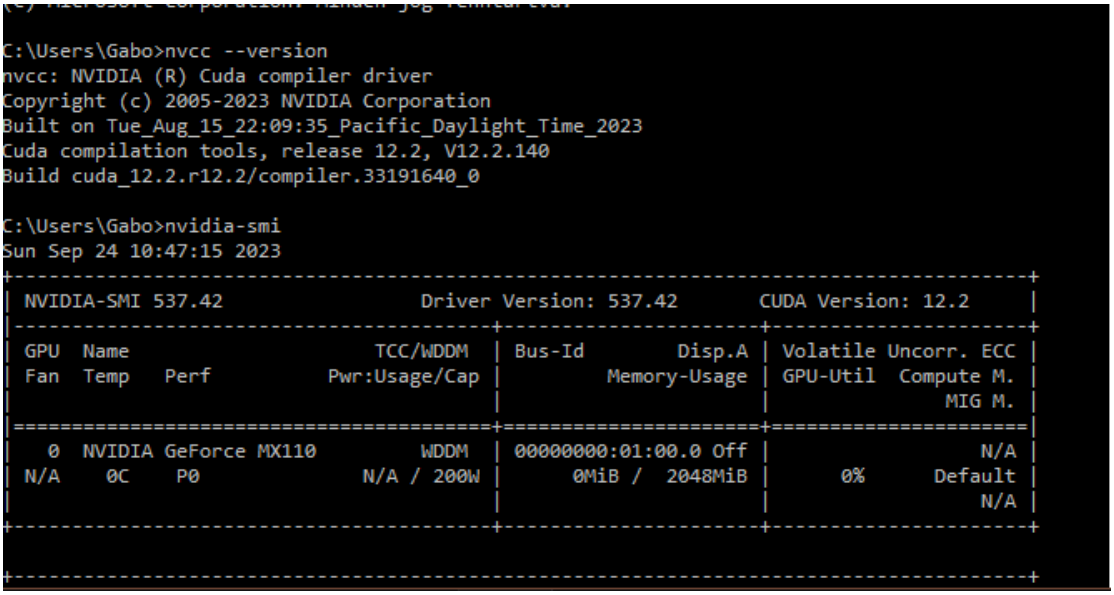
A 3 részre bontott kurzusban a Flan-T5 nevű sequence-tosequence, azaz kódolót és dekódolót is tartalmazó modellel végeztethetünk szövegösszefoglaló feladatot, a modell válaszainak helyességét pedig növelhetjük a különböző módszerekkel. Az első szakaszban az in context learning-et próbáltam ki. A várható eredményeket sikerült tapasztalnom, a few-shot inference adta a legjobb összegzéseket, míg példa nélkül nem igazán a kért feladatnak megfelelő szöveget kaptam vissza. Túl sok példánál is problémák léptek fel a túl sok kontextus miatt. Rövidebb és kevesebb szöveget megadva példaként már hatékonyabb volt.

A 2. részben a finomhangolást próbáltam ki. Itt már kezdett kiütközni a dolog erőforrásigényessége, a 3 éves középkategóriás laptopomon elég lassan történt meg egy-egy tanítás. Mivel alapesetben a tanítás a processzoron történik, és videokártyán ez sokkal gyorsabb, ezért a CPU-ról GPU-ra átállást választottam. Ehhez a CUDA-t kellett telepítenem. Ezt sikeresen megtettem a legfrissebb verzióval, parancssorban kiadva az

[19] Stanford University AI Lab igazgatója, https://en.wikipedia.org/wiki/Andrew_Ng

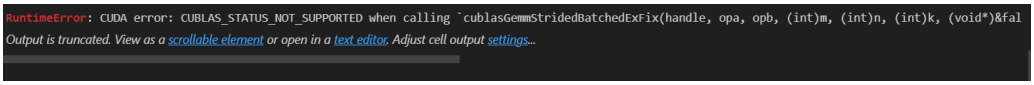
[20] Coursera kurzus linkje: https://www.coursera.org/learn/generative-ai-with-llms

nvcc –version és az nvidia-smi parancsokat meg is bizonyosodtam erről (4.1 ábra). A PyTorch viszont ezt nem érzékelte, a torch.cuda.is_available() függvény, amivel le lehet kérdezni, hogy elérhető-e a gépen a könyvtár, hamis értéket adott vissza. A probléma oka, hogy a PyTorch adott verziója, nem volt még kompatibilis a CUDA ezen verziójával. Ezt a gondot a torch csomag megfelelő telepítésével oldottam meg: *pip install torch==2.1.0+cu118 --index-url https://download.pytorch.org/whl/cu118*



4.1. ábra CUDA 12.2 telepítve

Így már el tudtam indítani a tanításokat a videokártyán, viszont futásidejű hibát kaptam:



4.2. ábra Futásidejű CUDA hiba

Sajnos a rengeteg kutatás után, hogy hogyan lehet orvosolni ezt a problémát, mi okozhatja, sem sikerült megoldanom. Emiatt arra a következtetésre jutottam, hogy az MTX 110-es laptop videokártyámon ezt nem fogom tudni megoldani, így egyelőre maradtam a CPU-n futtatásnál. A megoldási javaslatok kimerültek a driver és a könyvtár frissítésénél, amit meg is tettem volna, ha nem azt használtam volna. A PyTorch fórumon feltett kérdésemre pedig érkezett egy válasz, hogy küldjek el egy kódrészletet, elküldtem, nem reagáltak erre már semmit se.

Az utolsó szekcióban a modellt a kártékony válaszok csökkentésére taníthatjuk megerősítéses tanulással. Ezzel elérhetjük, hogy kevésbé generáljon sértő, toxikus

szöveget. Úgy gondoltam, hogy erre csak akkor lenne szükségem a chatbothoz, ha egy teljesen új modellt akarnék létrehozni. Ha a tanító adathalmazomban vagy a használt forrásokban, nincsenek ártalmasnak mondható megfogalmazások, akkor az előre tanított modell ilyen szintű beállítása elég kell, hogy legyen.

Ezek után két másik kurzust is végignéztem a DeepLearning.AI-tól. Ezek rövidebbek, mint az előző, és a LangChain könyvtár lehetőségeiről, használatáról szólnak, hogyan lehet vele nagy nyelvi modellekre épülő alkalmazásokat, chatbotokat építeni.[21] Mindkét esetben a használt nyelvi modell az OpenAI API-n keresztül elérhető GPT-3.5 Turbo, ami akár cserélhető GPT-4-re is, ha már van az embernek elérése ahhoz is. A keretrendszer nagyon sok LLM-es problémára nyújt egyszerű megoldást, ezért döntöttem úgy, hogy ezt fogom használni a chatbothoz majd elsősorban [15].

4.2 Modellek keresése

Miután megismerkedtem az alapokkal, első feladatnak azt tűztem ki, hogy olyan modellt találjak, ami kellően tud magyarul egy chatbot megvalósításához. A legjobb egy olyan modell lett volna, ami angolul is tud, mert a későbbiekben külföldi hallgatók is kérdezni tudnának tőle, de nem ez volt az elsődleges szempont.

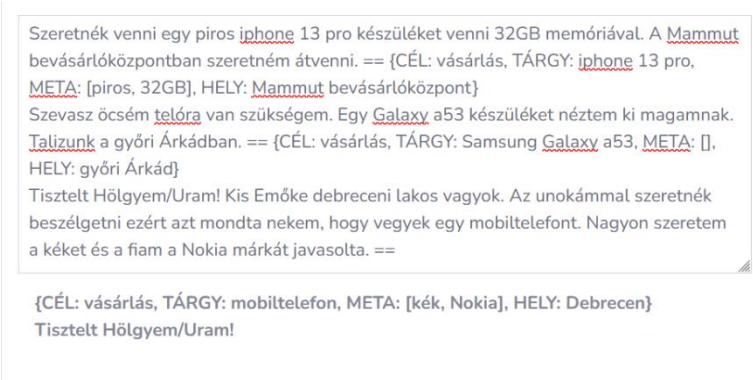
Az először magyar fejlesztésű modellekre kerestem, arra számítva, hogy a magyar nyelvhez jobbak lesznek ezek. A Nyelvtudományi Kutatóközpont (NyTK) a legnagyobb szervezet Magyarországon, amely foglalkozik nagy nyelvi modellek tanításával.

4.2.1 PULI GPT-3SX

A PULI GPT-3SX modellt az NyTK készítette 2022-ben. 6,7 milliárd paraméteres GPT-NeoX típusú (egy GPT-3-hoz hasonló felépítésű [16]) autoregresszív modell, tehát csak dekódert tartalmaz [17][18]. Kipróbálható[22] saját vagy pár javasolt prompttal is, többet is megnéztem. Azt tapasztaltam, hogy nagyjából jól válaszol a kapott szövegre, viszont sokszor plusz szöveget generál feleslegesen (4.3 ábra), vagy pont az ellenkezője, nem fejezi be a mondatot, félbehagyja azt.

[21] A két kurzus: https://learn.deeplearning.ai/langchain és https://learn.deeplearning.ai/langchain-chat-with-your-data

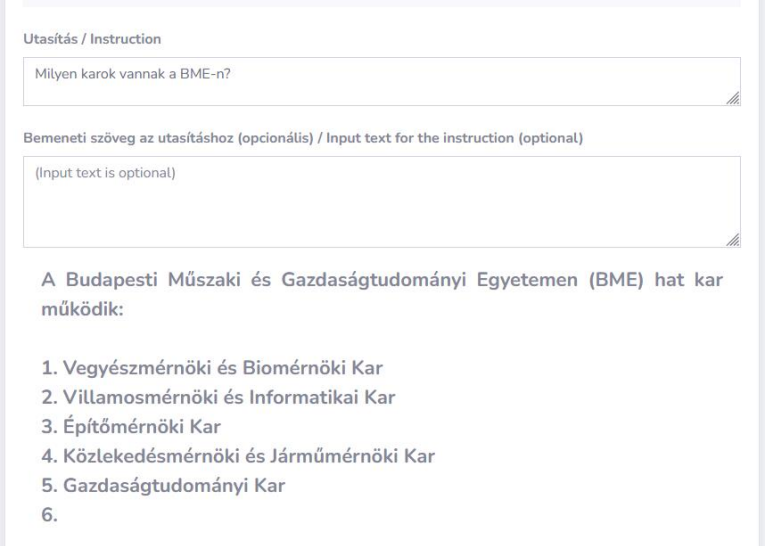
[22] https://juniper.nytud.hu/demo/puli



4.3. ábra PULI GPT-3SX példa prompttal

4.2.2 PULI-GPTrio

A PULI-GPTrio szintén egy csak dekódert tartalmazó modell az NyTK-tól. Ez is GPT-NeoX, de már 7,6 milliárd paraméteres. 2023 első felében tanították a kutatóközpontnál 3 nyelven: magyar, angol és kínai. Finomhangolt, Instruct LLM verziója online kipróbálható.[23] Eredményességben jobb az előző modellnél, jobban a kapott kérdésre, utasításra válaszol. Itt is előfordult azonban a félbehagyott mondat, válasz (4.4 ábra). Volt viszont olyan eset is, hogy egy húsleves receptet kértem tőle, amire szépen levezette az egészet, viszont érdekes hozzávalókat is javasolt beletenni, mint például tejet, illetve az elkészítési lépések között volt, hogy zsírban süssem meg a húst zöldségekkel, majd a leveshez adva botmixerrel pürésítsem.



4.4. ábra PULI-GPTrio válasza egy kérdésre

[23] https://juniper.nytud.hu/demo/parancspuli

A legjobb eredményeket meglátásom szerint ezekkel a paraméterekkel adta:

- Temperature: 0,7
- Top-p: 1
- Top-k: 20
- Maximális tokenszám: 500
- Repetition penalty (Ismétlés büntetése): 1,0. ennek állítása befolyásolta legjobban a generált szöveget, kisebb értéknél sokszor csak saját magát ismételgette csak a modell.

Ezeknek a modelleknek a futtatására lokálisan nem sikerült. Mindkettő modellnek több, mint 16 GB-nyi memóriára volt szüksége, így nem tudtam futtatni a laptopomon. A két PULI modellt a Hugging Face-es[24] transformers Python könyvtárral próbáltam beüzemelni.

4.2.3 Finomhangolt huBERT

Az előző két modell után a Hugging Face-n kerestem további magyar modelleket, melyeket a chatbothoz használhatnék. Ekkor találtam rá az mcsabai (Csabai Máté) felhasználó által kérdésmegválaszolásra finomhangolt huBERT modellre. Az alap huBERT a Számítástechnikai és Automatizálási Kutatóintézet (SZTAKI) által lett előre tanítva [19]. Csak enkódert tartalmazó modell, amit a Stanford Question Answering Dataset (SQuAD) 2.0[25] Google Fordító által lefordított kérdés-válasz adathalmazzal lett finomhangolva.

Ez a modell kicsit máshogy működik, mint az előzőek, mivel ez egy kérdés megválaszoló (question answering) modell. Külön meg kell adni neki a kérdést és a kontextust, amiből kiválasztja a választ rá. Egy-egy kérdésre jó válaszokat ad, de egy összetettebb, több dologra irányuló kérdésnél csak az egyikre válaszol, ahogy ez látható a 4.5-ös és a 4.6-os ábrán is. Kontextusnak a Python Wikipédia első bekezdését adtam meg, melyben a válaszok megtalálhatóak a „Ki találta ki a Pythont?” és a „Mi az a Python és ki találta ki?” kérdésekre is.

[24] NyTK oldalán megtalálható mindkettő: https://huggingface.co/NYTK

[25] https://rajpurkar.github.io/SQuAD-explorer/

```
Kérdés: Ki találta ki a Pythont?
Válasz: Guido van Rossum
Konfidencia: 0.9838784337043762
{'score': 0.9838784337043762, 'start': 76, 'end': 92, 'answer': 'Guido van Rossum'}
```

4.5. ábra huBERT egy kérdéssel

```
Kérdés: Mi az a Python és ki találta ki?
Válasz: Guido van Rossum
Konfidencia: 0.506941020488739
{'score': 0.506941020488739, 'start': 76, 'end': 92, 'answer': 'Guido van Rossum'}
```

4.6. ábra huBERT két dologra kérdezve

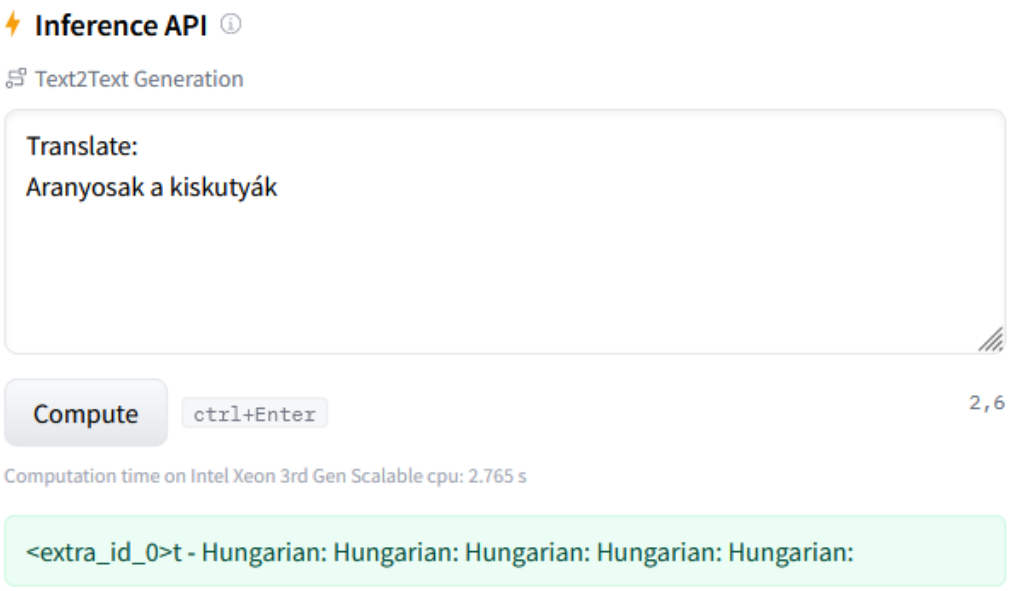
Látható, hogy a második esetben csak az utóbbi kérdésre válaszol, viszont majdnem feleannyira biztos a válaszában. Kisebb erőforrásigénye miatt lokálisan is tudtam futtatni a modellt. Egy kérdésre tanúsított jó válaszai miatt egy egyszerűbb, csak válaszolni tudó, de igazi párbeszédre nem alkalmas chatbothoz megfelelőnek találtam.

4.2.4 mT5

Az első, hosszabb kurzusban használt FLAN-T5 nevű modell sajnos nem alkalmas magyar nyelvű kommunikációra, viszont a Google a T5 modelljét az mC4 korpuszt[26] felhasználva többnyelvűre tanította azt 2020-ban. Így született meg az mT5, mely 101 nyelven képes szöveget generálni [20]. Text-to-Text modell, tehát enkódert és dekódert is tartalmaz. Eredményeit tekintve nem igazán hasznos további tanítás nélkül (4.7 ábra), de ezt írják is a Hugging Face-s oldalán[27], hogy további finomhangolás szükséges.

[26] https://huggingface.co/datasets/mc4

[27] https://huggingface.co/google/mt5-base



4.7. ábra mT5 kipróbálása

4.2.5 Llama-2

A Meta 2023 nyarán bemutatta új LLM-családját, a Llama-2-t [21]. Többféle paraméterszámú modellt, illetve alap és chatre optimalizált változatokat is elérhetővé tettek. Csak dekóderrel rendelkeznek. Bár nyílt forráskódúnak van említve a marketinganyagokban, nem teljesen az. A modelleket lehet használni, de csak elérés igénylése után, és csak akkor ingyenesen, ha programunk kevesebb, mint 700 millió havi felhasználóval rendelkezik, más modellek tanításához szükséges adatok generálása se lehet használni. Ezzel valószínűleg a konkurenciát (főleg a Google-t) akarták kizárni. Sokáig nem kaptam meg az elérést, így a félév során nem sok időm volt ezekkel a modellekkel foglalkozni. A chat változatok teljesítményét a ChatGPT-vel összemérhető szinten említik.

4.2.6 mGPT

A mGPT a GPT-3 egyik többnyelvű változata. 25 nyelvcsaládból összesen 61 nyelvre tanították, köztük magyarra is.[28] A tanító adathalmazok a Wikipédia és az mC4 voltak. Képességeiben a magyar modellekhez hasonlónak találtam, abban a tekintetben, hogy nem generálja le a teljes válaszokat (4.8 ábra) [22].

[28] https://huggingface.co/ai-forever/mGPT

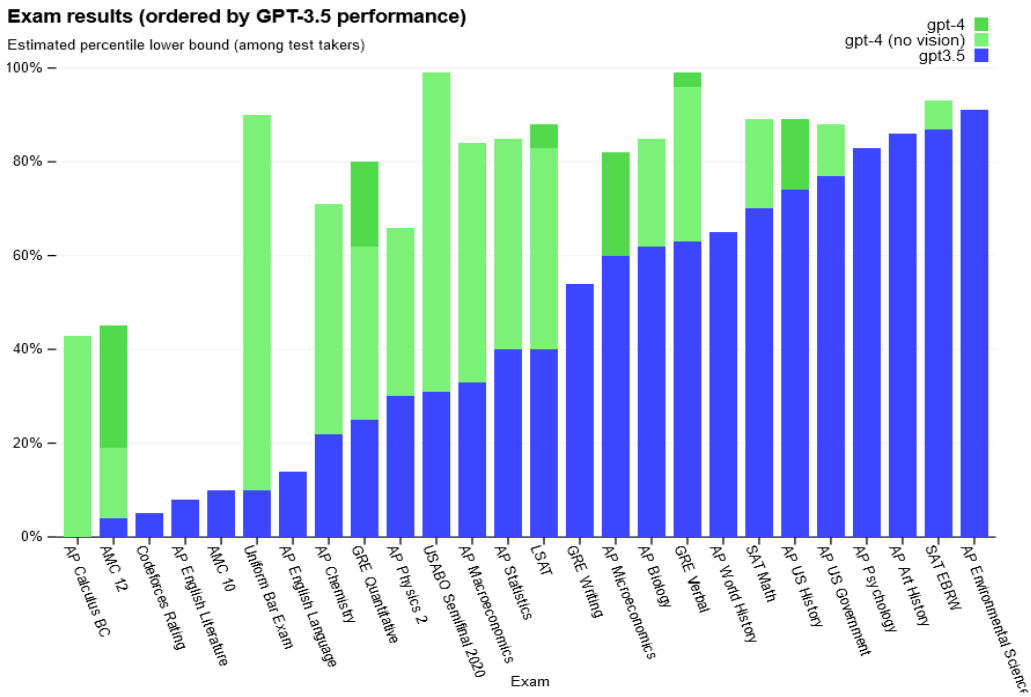
4.8. ábra mGPT generált szöveg megszakad

4.2.7 GPT-3.5 Turbo és GPT-4

A GPT-3.5 Turbo és GPT-4 az OpenAI két fő modellje a dolgozat írásakor, minkettő csak dekódert tartalmaz. Előbbi alapverzióját 2022 elején tanították, majd a ChatGPT-ben vált elérhetővé a chatre optimalizált változata az év novemberében. 2023 márciusában fejlesztőknek is elérhetővé vált az OpenAI API-n keresztül, így akár saját maguk által készített programokhoz is fel tudták már azt használni. Több vállalat le is csapott egyből rá, mint például a Snapchat (amerikai multimédiás csevegőalkalmazás), ahol My AI néven testreszabható chatbottal beszélgethetünk, vagy a Quizlet, ami egy mesterséges intelligenciával támogatott tanulóplatform [23]. A modell Codex nevű változata adja a GitHub Copilot fejlesztőkörnyezetbe implementálható, programozókat segítő bot alapját.

A GPT-4-et szintén márciusban jelentették be, melyet azóta is ChatGPT Plus felhasználók használhatnak csak az online felületen. A modell kreatívabb, pontosabb válaszokat ad elődjénél. Hatalmas újítása, hogy nem csak szöveges, hanem vizuális inputot is megadhatunk neki. Ennek is köszönhető, hogy bizonyos felméréseken már emberi szinten képes teljesíteni (4.9 ábra). 2023 júliusától az API-n keresztül is általánosan elérhető azoknak, akik már egyszer használták annyit az API-t, hogy fizetniük kelljen. Ezt a modellt használja bizonyos funkcióihoz például a Duolingo, a Copilot X (a Copilot fejletteb verziója) is. A Be My Eyes nevű applikáció a vakoknak és a gyengénlátóknak a mobiltelefon kameráján keresztül segít a „látásban” a vizuális

bemenetet használva. A Microsoft is a saját keresőjét, a Binget, a GPT-4 erejével emelte új szintre.



4.9. ábra A két OpenAI modell teljesítménye, forrás: [24]

Az OpenAI szolgáltatásainak magyar tudása nagyon jó, válaszolni is teljes mondatokban tud. Hátrányaihoz sorolható, hogy nem tudjuk a modellt saját kézben kezelni, a vállalat szervereire vagyunk bízva ilyen szempontból, illetve az is, hogy az API használata egy kis időszak vagy kb. 20 dollárnyi adatforgalom után már fizetős. Szerencsére nem havidíjas, hanem a használat után, input és output tokenekkel arányosan kell fizetni érte. Már a ChatGPT online verziója (ami ingyenes) is regisztrációhoz kötött.

Az önálló laboratórium témám, illetve a szakmai gyakorlaton végzett munkám révén már jól megismertem az OpenAI keretrendszerét, ezért úgy határoztam, hogy a chatbotomat először erre fogom építeni, és egy megfelelő minőségű alternatíva esetén fogom a modellt leváltani.

4.3 Adatok megkeresése és előkészítése

A modellek keresése után az adatok összegyűjtését, rendezését és megfelelő formátumba szervezését tekintettem a következő lépcsőnek. Az adatok feldolgozásához egy Jupyter notebook-ot hoztam létre process_data.ipynb néven. Első körben az egyetem

szakmai gyakorlat szabályzatát[29] töltöttem le. Átkonvertáltam txt fájlba PDF-ből a könnyebb feldolgozás, szerkeszthetőség érdekében és töröltem belőle a kérdésmegválaszolás szempontjából felesleges részeket, mint például a tartalomjegyzéket és a mellékleteket.

A következő hely, ahonnan információkat szereztem, az Automatizálási és Alkalmazott Informatikai Tanszék (AUT) szakmai gyakorlat portálja[30] volt. A gyakran ismétlődő kérdések (GyIK) oldallal kezdtem. A HTML kódot megvizsgálva feltűnt, hogy mindegyik kérdést tartalmazó <p> taghez a question osztály van rendelve, így a Beautiful Soup könyvtárat használva az összes kérdést és mindegyikhez a következő tagben szereplő választ egy pandas DataFrame-be raktam, majd lementettem a gyik.csv fájlba. A portál főoldaláról és a VIK-es tudnivalók oldalról[31] is szöveges formátumban mentettem le a main role-lal vagy id-val rendelkező, tényleges információkat tartalmazó részeket. Az AUT-os portál teendők oldaláról pedig a táblázatot raktam egy CSV-be.

Ezek után ki kellett találnom, hogyan lenne legjobb ezeket felhasználni a chatbothoz. Úgy ítéltem meg, hogy ezeken az a forrásokon felül még jó lenne, ha még több kérdés-válasz pár lenne, nem csak a gyik.csv-ben szereplők. Ezekből könnyen ki lehet következtetni mi egy-egy kapott inputra a megfelelő válasz. A számos GyIK-szerű kérdés megfogalmazása azonban rengeteg időt venne igénybe, így ehhez is a nagy nyelvi modelleket vettem segítségül. Az OpenAI API-t használva a szöveges fájlokból extra kérdésbankot generáltattam. Először a GPT-3.5, majd a GPT-4-es modellel is elvégeztem ezt, majd a kapott több száz kérdést egy-egy CSV fájlba mentettem. Mivel nem lehettem biztos benne, hogy valóban jó és értelmes kérdés-válasz párok generálódtak, ezért manuálisan végignéztem a két gyűjteményt. A szelektált kérdésekkel teli fájlokat pedig a használni kívánt többi mellé tettem.

A tárgyfelelőstől megkérdeztem, ezeken az oldalakon felül vannak-e egyéb források, illetve milyen kérdésekkel szoktak még hozzá fordulni, amik kicsit

---

[29] https://www.vik.bme.hu/document/412/original/Szakmai_gyak_BSc_szab%C3%A1lyzat_2014%20ut%C3%A1n.pdf

[30] https://www.aut.bme.hu/SzakmaiGyakorlat

[31] https://www.vik.bme.hu/kepzes/gyakorlat/442.html

specifikusabbak. Kaptam tőle 200 emailt, hogy felhasználhassam a chatbot adatbázisának növelésére. Mivel bizalmas információkat tartalmazhatnak ezek a levelezések, így egy az egyben nem lehet a források közé illeszteni őket. Azzal kezdtem az emailek átalakítását, hogy a személyes információkat kiszedjem belőlük. Ezt bizonyos szavak, szókapcsolatok meghatározásával tettem meg.

Elsőként az emailekre vonatkozó kifejezéseket határoztam meg: „From:”, „To:”, „Cc:”, „Sent:”, (ugyanezek magyarul is, mert pár levelezőrendszer úgy használja). A levél tárgyát meghagytam, úgy gondoltam az hasznosabb lehet, ha marad. Következő a köszönés-elköszönés volt, aztán olyan infók, mint például a Neptun-kód. A maradék szövegeket megvizsgálva láttam, hogy több cégtől is írtak neki, így a cégadatokat (adószám, számlaszám, cím, telefonszám stb.) is megpróbáltam kiszűrni. Ezeken felül a vírusirtókra vonatkozó szövegrészeket is töröltem, mert ezek zavarták az értelmezést. Az így megmaradt szövegeket a korábbi módhoz nagyon hasonlóan az OpenAI API segítségével, kérdés-válaszokká formáltam. Keletkezett körülbelül 800 elem, ezeket szintén manuálisan átnéztem. A rengeteg ismétlődés, pár értelmetlen vagy még így is bizalmas információt tartalmazó kérdéseket kiszűrve kb. 230 elemmel gyarapíthattam a használatra alkalmas adathalmazt.

Az így kialakított adatok nagy része tehát kérdés-válasz párokban volt megfogalmazva, ami finomhangoláshoz is megfelelő lehet.

4.4 Chatbot felépítése

A chatbot felépítésének átgondolása volt a következő lépés a projektben. Az természetesen egyből egyértelmű volt, hogy magában egy nagy nyelvi modell nem lesz elég, hiszen az nem tudja önmagában a chat előzményeit tárolni. A dokumentumokat is azért gyűjtöttem össze, mert egy LLM nem tudja ezeket a specifikus információkat.

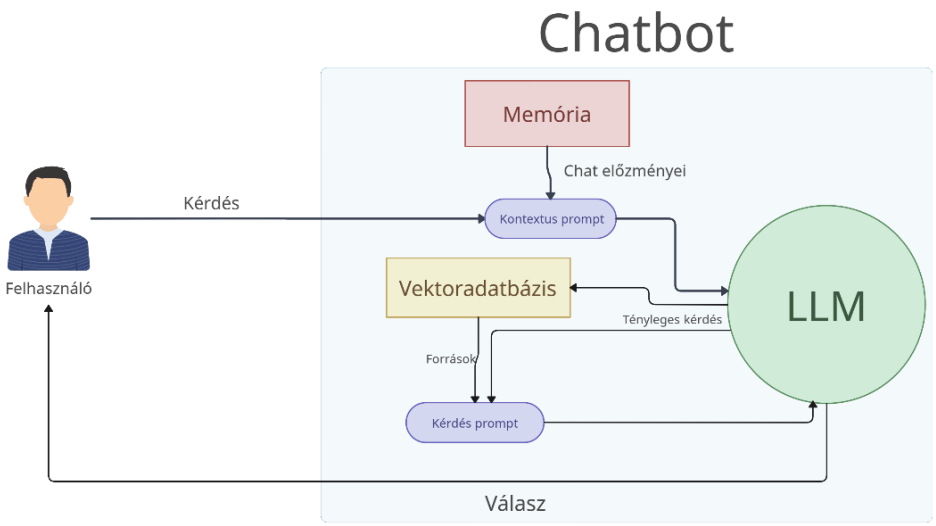
Utóbbira van két opció is: finomhangolni és RAG architektúrával adatbázisból kinyerni az adatokat. Mivel a finomhangoláshoz rengeteg adatra lehet szükség, hogy ténylegesen működjön is, mindenképp egy vektoradatbázist is terveztem a botba rakni.

Memóriára szükség van, mert az előzmények nélkül lehetnek esetek, mikor egy-egy kérdést nem tudunk értelmezni, nincs kontextusa.

Például:

- **Felhasználó**: Ki a tárgyfelelős?
    - **Bot**: Dr. Blázovics László
- **Felhasználó**: Hol tudom őt elérni?
    - **Bot**: ???

A fenti esetben, ha nem tudjuk mi volt az előző kérdés, fogalmunk sincs, ki az ő, akit el szeretnénk érni, így a források keresése se biztos, hogy jó eredményre fog jutni. Ha megnézzük a korábbi üzenetváltást, akkor egyből egyértelmű, hogy kire vonatkozik.



4.10. ábra A chatbot tervezett felépítése

A 4.10-es ábrán felvázoltam a chatbot működését. A felhasználó egy kérdést, üzenetet fogalmaz meg és elküldi a botnak. Az üzenete először egy olyan promptba kerül a chat előzményeivel együtt, amely arra való, hogy egy önálló kérdést is meg tudjunk fogalmazni a kontextus alapján, ha szükséges. Ezután ezt a promptot megkapja a nyelvi modell és megfogalmazza a konkrét kérdést. A korábbi példánál ez egy olyan kérdés lehet, hogy „Hol tudom elérni Dr. Blázovics Lászlót?”. Ha a felhasználónak az első üzenetét kapjuk meg, tehát üres a memória, akkor ez a lépés kihagyható.

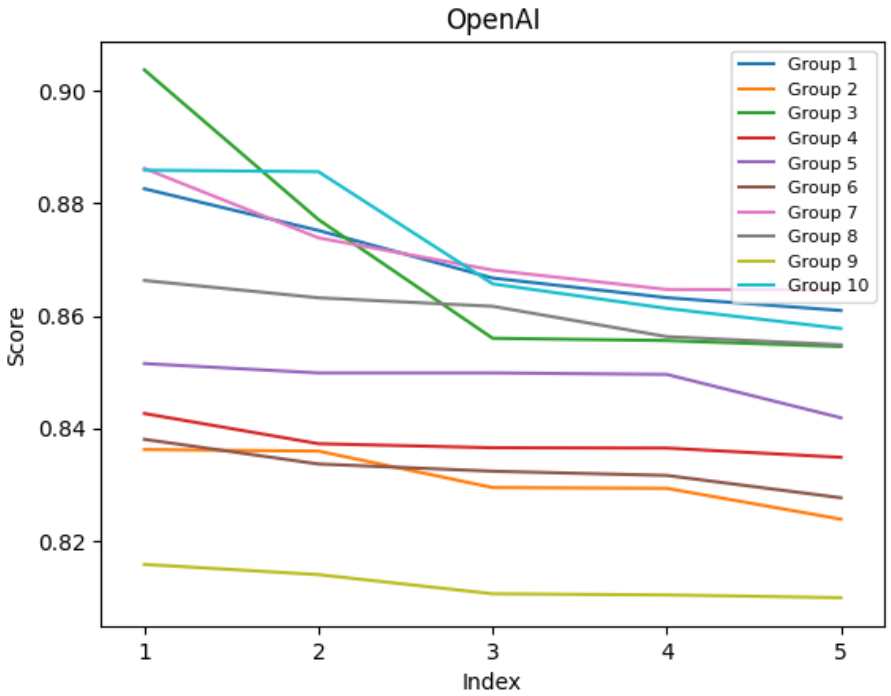
Az üzenetre ezután rákeresünk a vektoradatbázisban, ugyanazzal az embeddinggel vektorra képezzük le és a vektorok távolsága alapján egy vagy több forrást kiválasztunk az adatbázisból. A forrás(oka)t aztán a kérdéssel együtt egy újabb promtpba rakjuk, aminek az a célja, hogy a kapott információk alapján válaszoljon a kérdésre az LLM.

4.5 Embedding kiválasztása

A vektorokra leképezéshez egy olyan embedding kell, amely megfelelően tud egy-egy szövegrészletet kódolni számértékekké, majd később tényleg a kapott szöveghez hasonló jelentésű forrásokat találja meg. Természetesen itt is vannak különféle opciók, így ezeket is meg kellett vizsgálnom. Háromféle lehetőséget vizsgáltam meg. 10 kérdést[32] adtam meg, mindegyiknél az 5 legközelebb lévő vektorhoz tartozó dokumentumot és ahhoz tartozó relevanciaértéket vizsgáltam meg. A teszteknél a szöveges forrásokat minden esetben maximum 500 token méretűre daraboltam. Mindegyikhez készítettem Matplotlibbel grafikont az eredmények szemléltetésére.

4.5.1 OpenAI Embedding

A GPT modellek használata mellett számos más lehetőség van az OpenAI API-n keresztül. Egy ilyen, a cég embedding modelljeinek a használata, melyek közül a legjobb a text-embedding-ada-002 nevű, így ezt használtam. Ez többféle nyelv leképezésére is alkalmas. Mint minden OpenAI API szolgáltatás, ez is fizetős.



4.11. ábra OpenAI Embedding eredmények

[32] a testing mappa test_questions.csv fájljában megtalálhatók

A 4.11-es ábrán látható, hogy a relevancia értékek szempontjából 0.8 feletti értékeket ér el. A tényleges eredményeket megnézve látható, hogy jó eredményeket is ad, ahogy az várható volt. Az alábbi táblázatban látható az 5 legjobb értékű találat.

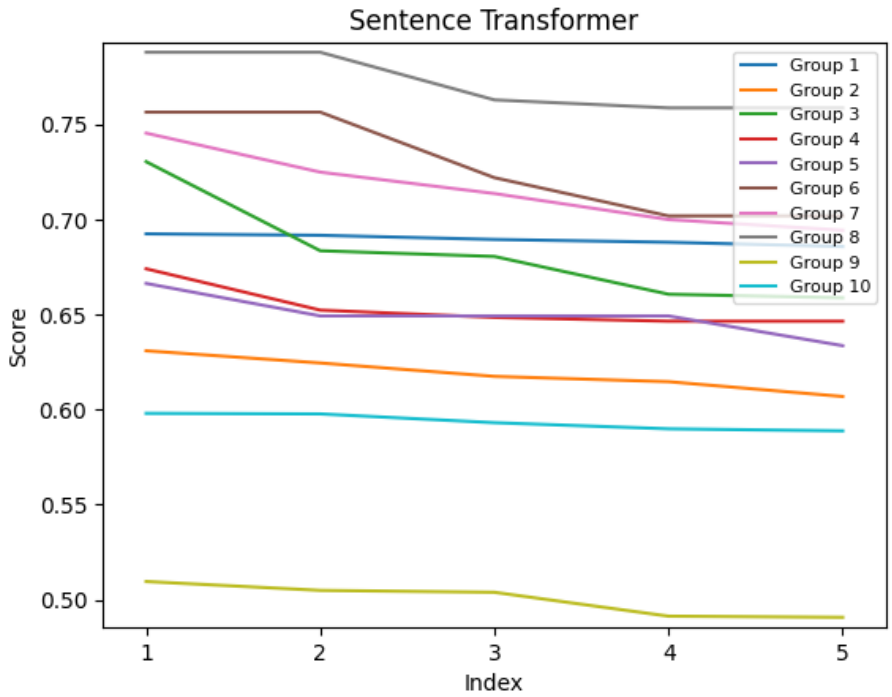
Kérdés	Dokumentum	Relevancia érték
Ki a tanszéki felelős?	Question: Ki a tanszéki felelős? Answer: A tanszéki felelős Dr. Blázovics László.	0.904
Van szabály arra, hogy milyen témájú gyakorlatot kell választani?	Question: Hány témát lehet választani a gyakorlat során? Answer: A gyakorlat során a hallgató legfeljebb 5 témát választhat, amelyeket a tanszéki felelős vagy közvetlenül a gyakorlat során tervezett munkákhoz rendel.	0.886
Van szabály arra, hogy milyen témájú gyakorlatot kell választani?	Question: Hogyan válasszunk témát a szakmai gyakorlatra? Answer: A szakmai gyakorlat témáját a hallgató választhatja meg, figyelembe véve a képzési szintjét, a specializációját és a gyakorlóhely ajánlásait.	0.886
Félév közben is el lehet kezdeni a szakmai gyakorlatot?	Question: Milyen időpontban lehet kezdeni a szakmai gyakorlatot? Answer: A javasolt kezdési időpont a vizsgaidőszak vége, de a szakmai gyakorlat a szabályzatban foglalt időpontok szerint végezhető.	0.883
Ki a tanszéki felelős?	Question: Ki a tanszéki felelős a szakmai gyakorlattal kapcsolatban? Answer: Dr. Blázovics László a tanszéki felelős.	0.877

4.5.2 Sentence transfromer

A sentence transformer-ek olyan modellek, melyek kifejezetten mondatok vektorokká alakítására lettek kifejlesztve [25]. Tartozik hozzájuk egy Python könyvtár[33] is, de Hugging Face-n keresztül[34] is használhatók. A modellek között van több is, amely magyarul is tud, ezek közül választottam a *distiluse-base-multilingual-cased-v2*-t. Ez a modell 512 dimenziójú vektorokra képezi le a kapott bemenetet.

[33] https://www.sbert.net/

[34] https://huggingface.co/sentence-transformers



4.12. ábra Sentence transformer eredmények

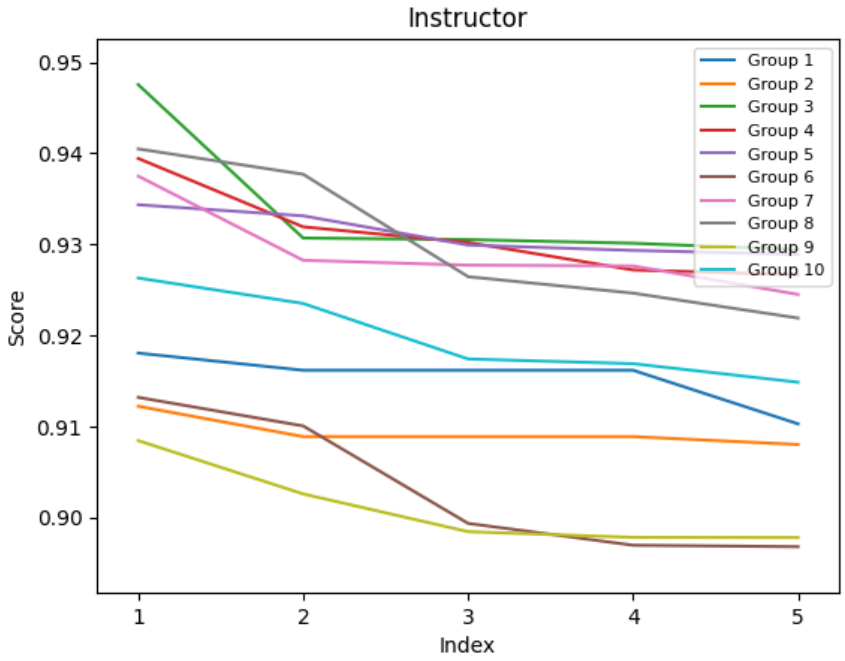
Ahogy a 4.12-es ábrán is látható, a legjobb értékek sokkal alacsonyabbak, mint az OpenAI-nál, 0,8-at egyik sem éri el. A táblázatból kiolvasható, hogy a tényleges információtartalma is elég gyenge az előző modellhez képest.

Kérdés	Dokumentum	Relevancia érték
Tárgynak felel meg a szakmai gyakorlat?	A szakmai gyakorlat szempontjából releváns tevékenységi köre:	0.788
Tárgynak felel meg a szakmai gyakorlat?	Gyakorlóhely esetén a szakmai gya korlat szempontjából releváns tevékenységi kör: .... Budapest, 201…….. …… Hallgató	0.763
Tárgynak felel meg a szakmai gyakorlat?	A szakmai gyakorlat témájának címe:	0.759

4.5.3 Instructor

A Hongkongi Egyetem, Washingtoni Egyetem, a Meta és az Allen Institute mesterséges intelligencia kutatói fejlesztették ki ezt a módszert [26]. Az Instructor egy utasításokkal finomhangolt embedding modell, ami különböző feladatokra tudja az utasítás szerint a szöveget leképezni. Lehet ez klasszifikáció, klaszterezés (csoportosítás), szemantikai keresés stb.[35]

Ehhez is tartozik egy Python csomag, de ugyanúgy Hugging Face-n is megtalálható. Az alap modellt[36] használtam a teszteléshez, és bár elvileg csak angol adatokon lett tanítva, meglepően jó eredményeket adott. Utasításként azt adatam meg, hogy dokumentumok kereséséhez alakítson ki vektorokat.



ű

4.13. ábra Instructor embedding eredmények

A 4.13-as ábra azt mutatja, hogy szinte az összes érték 0,9 felett van. E modell tesztelése közben jöttem rá, hogy az értékek nem feltétlen mérvadóak, hiszen a vektorok távolsága a saját számítása alapján lehetnek közelebbiek. Itt mégis azt látjuk, hogy pár eredménynek van köze a kérdésben lévő szavakhoz, így egy valós alternatíva lehet a

[35] https://instructor-embedding.github.io/

[36] https://huggingface.co/hkunlp/instructor-base

sentence transformerek helyett egy offline megoldáshoz annak ellenére, hogy elvileg csak angolul tud.

Kérdés	Dokumentum	Relevancia érték
Ki a tanszéki felelős?	Question: Ki a tanszéki felelős? Answer: A tanszéki felelős Dr. Blázovics László.	0.948
Tárgynak felel meg a szakmai gyakorlat?	A szakmai gyakorlat témájának címe:	0.94
Mennyit kell dolgozni a gyakorlat elfogadásához?	Question: Mikor kell felvenni a nyári gyakorlati munkához a tárgyat? Answer: Nyári gyakorlati munka esetén az őszi félévben.	0.939
Tárgynak felel meg a szakmai gyakorlat?	A szakmai gyakorlat témájának címe:	0.938
Mikor kell leadni a szakmai gyakorlati beszámolót?	Question: Mikor kell leadni a kérelmet? Answer: A kérelmet a szakmai gyakorlat megkezdése előtt kell leadni.	0.937

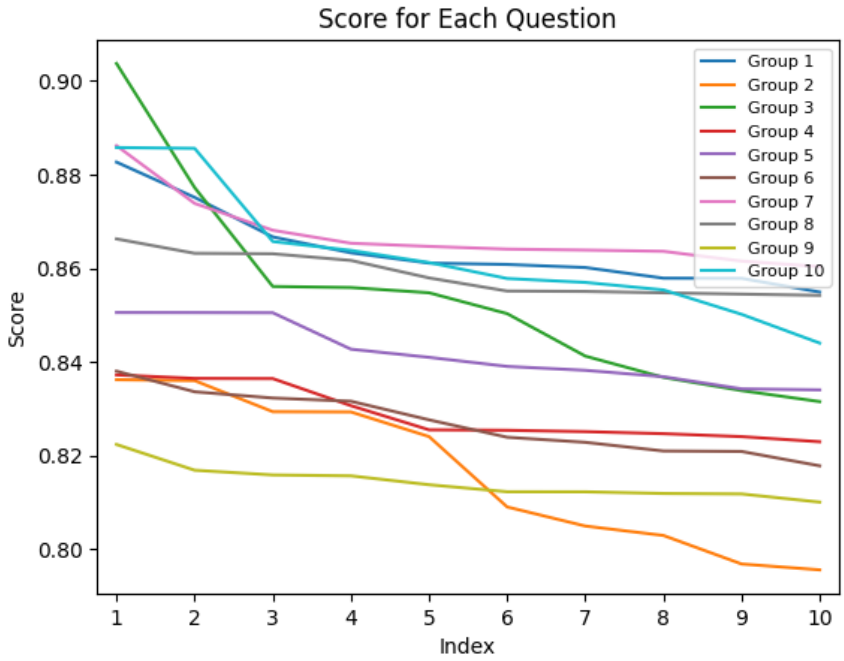
4.6 Paraméterek meghatározása

A keresés és a szövegdarabolás paramétereinek meghatározásához hasonló módszert alkalmaztam, mint az embeddingeknél. A konzisztens eredmények érdekében itt végig az OpenAI embeddinget használtam, a kérdések pedig ugyanazok maradtak, mint a korábbi teszteknél. A 10 legjobb értéket mutató grafikonokon (4.14 és 4.15 ábra) látszanak az eredmények.

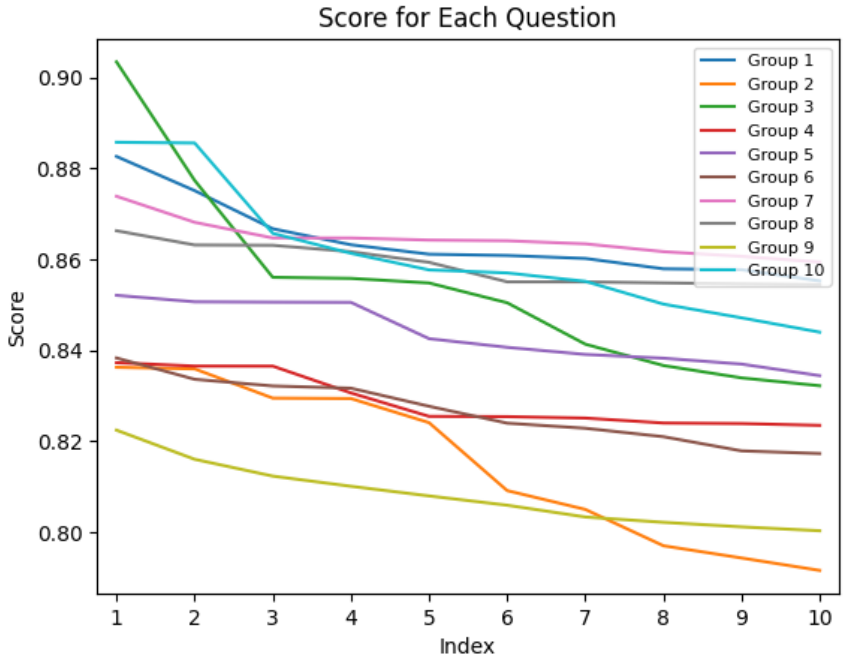
Legjobb értékhez tartozó kérdés (Group 3): „Ki a tárgyfelelős?”. Gyors esés tapasztalható azonban, mert nagyon specifikus és nincs túl sok plusz információ ehhez még.

Átlagosan legrosszabb értékű kérdés (Group 9): „Van szabály arra, hogy mennyit kell dolgozni?”.

Legrosszabb értékű kérdés (Group 2): „Félév közben mennyit szabad hetente dolgozni?”. Erre vannak konkrét információk, ezért jobb értékei vannak az elején, de aztán nagyon gyorsan már nincs igazán köze a kérdéshez a dokumentumoknak.



4.14. ábra 10 legjobb elem max 250 hossznál



4.15. ábra 10 legjobb elem 500 max hossznál

A 4.14-es ábrán látható az, amikor a maximum hosszúság 250 volt, a 4.15-ösön pedig az, amikor 500. Látszik, hogy nem nagyon változnak a grafikonok. Erre az a magyarázat, hogy sokszor a gyakori kérdésekben talál megfelelő információkra, amik hosszán nem változtat ez a paraméter. Leolvasható, hogy a 8. elem után az esetek többségénél eléggé csökken a relevanciája egy-egy forrásnak, így az MMR kereséshez a fetch_k-t, azaz amennyiből kiválasztja a k darab dokumentumot, 8-ra állítottam. A k-t

pedig 5-re, hogy legyen benne elég mennyiségű forrás a kérdés megválaszolásához. Az adatbázis bővítése esetén ezeket a számokat érdemes felülvizsgálni.

4.7 Teams integráció

Egy chatbot használatához Teams-ből két módját találtam microsoftos források alapján

4.7.1 Power Virtual Agents[37]

Ez az újabb lehetőség, azonban sokkal limitáltabb megoldásnak tűnik. Botok készítéséhez folyamatábra-szerűen rakosgathatunk össze különböző szituációkat: milyen szókapcsolatokra, válaszokra hogyan reagáljon, mit hajtson végre. Ebben hasonlít a Bot Framework Composerre. Saját kód futtatására nem találtam lehetőséget.

4.7.2 Microsft Bot Framework

A Microsoft Bot Framework egy keretrendszer, amely elérhető .NET-ből, JavaScriptből és Pythonból is. Tényleges kódot is tudunk írni, a botot Azure-ben kell aztán publikálni. Megfelől beállításokkal a botunkat be tudjuk építeni Teams-be, Facebook-hoz, illetve email címhez is köthetjük stb. A chatbotomhoz ezt választottam.

[37] https://powervirtualagents.microsoft.com/en-in/

5 A chatbot megvalósítása

A chatbot megvalósítását azzal kezdtem, hogy VS Code-ban létrehoztam egy Jupyter notebookot, és ebben elkezdtem összerakni a bot építőelemeit. A kód első felében a paraméterek gyűjtöttem össze, egyrészt azért, hogy egy helyen legyenek, másrészt azért, hogy ne kelljen változtatás esetén keresgélni, vagy több helyen is átírni az értékeket. Az ezt követő lépések: információk betöltése, adatbázisba szervezése, chat memória létrehozása, ezek összekötése a nyelvi modellel, a rendszer tesztelése, Azure-be telepítése, majd Teams-es elérés megvalósítása.

5.1 Adatok betöltése és feldarabolása

Az információforrások betöltéséhez a LangChain keretrendszer documents_loaders csomagját használtam. A res mappán belül az in_use-ba raktam az összes használni kívánt adatfájlt. A programban először a CSV fájlokat töltöm be, majd a txt fájlokat. Mindkét esetben egy DirectoryLoader objektumot használtam, amelynek meg lehet adni, hogy milyen típusú Loader osztályt használva melyik mappából, milyen nevű fájlokat töltsön be. Így néz ki a CSV fájlok betöltése:

```
directory_loader = DirectoryLoader(res_folder, glob="*.csv",
use_multithreading=True, loader_cls=CSVLoader, loader_kwargs={"encoding":
"utf-8"})
csv_data = directory_loader.load()
```

Így a csv_data-ban egy tömbben vannak a Document objektumok. Minden egyes sorból egy darab lett leképezve. Szöveges fájloknál a glob *.txt, a loader_cls értéke pedig TextLoader. Ez egy fájlból egy dokumentumot készít, ezért még fel kellett darabolnom kisebb szöveges részekre. Ehhez a RecursiveCharacterTextSplittert használom, ami rekurzívan vágja szét a szöveget a szeparátorok mentén, először az elsőként megadott szerint, majd ezeket a Documenteket is a következő szerint. A chunk_size-nak (maximum méretnek) 500-at, az átfedésnek (overlap) 50-et adtam meg. A két dokumentumlistát ezek után egybefűztem.

5.2 Adatbázis létrehozása

A következő teendőm az volt, hogy létrehoztam egy OpenAIEmbeddings objektumot. Ez is integrálva van a LangChainbe, az embeddings-en belül. Ezzel megvolt minden elemem, hogy létrehozhassam a vektoradatbázist a forrásokból, amiben aztán

kereshetünk. Ehhez a LangChain vectorstores moduljában számtalan lehetőség elérhető, én a Chroma-t használtam.

```
oai_embedding = OpenAIEmbeddings()
vectordb = Chroma.from_documents(
     documents=combined_data,
     embedding=oai_embedding,
     persist_directory=persist_directory
)
vectordb.persist()
```

A kódban az a verzió látható, mikor létrehozzuk az adatbázist dokumentumokból. A persist_directory paraméterben megadhatjuk a mentés helyét, ami az én esetemben a res/chroma mappa. A persist() függvény menti el ténylegesen a háttértárra. Ez a függvény az objektum elpusztulásakor is automatikusan lefut, de a biztonság kedvéért egyből mentem az adatbázist a háttértárra, hogy bármilyen váratlan hiba esetén meglegyen.

Fontos, hogy ha már van mentett adatbázisunk, ne így hozzuk létre a vectordb objektumot. Így képezve az adatbázist azt tapasztaltam, hogy egyre kevésbe lesznek változatosak a megtalált dokumentumok, sőt volt, hogy ugyanaz került bele a promptba többször is. Ennek oka az volt, hogy ez a megoldás a persist_directory-ban létező vektoradatbázisba belerakja documents paraméterben kapott lista elemeit is. Ha már egyszer lementettük, akkor a következő alkalommal már csak a konstruktorral hozzuk létre a vectordb-t:

```
oai_embedding = OpenAIEmbeddings()
vectordb = Chroma(
     persist_directory=persist_directory,
     embedding_function=oai_embedding
)
```

Az embeddingek teszteléséhez is el kellett ezeket végeznem, így azokat is beleraktam ebbe a notebookba.

5.3 Memória létrehozása

Egy memória létrehozása volt a következő építőelem a chatbot elkészítésében. Ehhez a ConversationBufferWindowMemory-t importáltam a langchain.memory-ból és azt használtam fel. Ez a típusú memória az előző k darab üzenetváltást tárolja el. K-nak hármat választottam, körülbelül ennyi kérdéssel korábbra utalnak vissza maximum az emberek.

```
memory = ConversationBufferWindowMemory(
      k = memory_k,
      memory_key="chat_history",
      return_messages=True
)
```

A memory_key segítségével megmondható, hogy egy prompt sablonban melyik változó helyére kerüljön a memória tartalma. A return_messages pedig azt mondja meg, hogy BaseMessage típusú elemekkel teli listát adjon vissza a buffer lekérdezése esetén. Ha False ennek a paraméternek az értéke, akkor stringet adott vissza, ezért hibát kaptam később a szöveg generálásakor, mert a sablonba nem tudta belerakni az előzményeket.

5.4 Nyelvi modellhez kapcsolás

Ezeket az egységeket össze kellett kapcsolnom. Ehhez számos beépített lehetőséget is biztosít a chains modul a LangChainen belül. Ezek közül a ConversationalRetrievalChain valósítja meg azt a működést, amire szükségem volt. Ennek használatához még egy LLM objektumot is létre kellett hozni, ebben az esetben egy ChatOpenAI típusú.

```
llm = ChatOpenAI(model_name= model_id, temperature = temperature, max_tokens
= max_tokens)
chain = ConversationalRetrievalChain.from_llm(
    llm,
    retriever = vectordb.as_retriever(
        searh_type = search_type,
        search_kwargs = {
            "k": search_k,
            "fetch_k": search_fetch_k,
            "lambda_mult": lambda_mult
        }
    ),
    verbose = True,
    memory = memory
)
```

A retriever paraméterben megadtam, hogy a Chroma adatbázisból maximal marginal relevance (MMR) kereséssel nyerje ki az adatokat. A lambda_mult egy 0 és 1 közötti szám, ami minél nagyobb, annál kevésbé változatosan fogja a fetch_k darabból kiválasztani a k darab elemet. Utóbbi kettőt a már taglalt 8-ra és 5-re vettem, a lambda_mult-ot pedig 0,6-ra, hogy kicsit a jobb értékűek felé billenjen a mérleg nyelve. A verbose érték True-ra állítása a debugolást könnyíti meg, ugyanis, így a standard kimenetre kiírja, épp hol tart, illetve milyen promptokat generált a sablonokból és az adatokból.

Ezek után már egy szimpla hívással, az üzenetet megadva megkaphatjuk a választ, ami egy JSON objektumot add vissza, az answer mezőjében van a tényleges válasz.

```
chain({"question": „Mennyit kell dolgozni?”})
```

Gradioval egy grafikus interfészt hoztam létre, majd manuálisan tesztelni, hogyan működik az elkészített chatbot. Ezalatt feltűnt, hogy néha angolul válaszol, hiába magyar minden forrás. Ez azért történt, mert a LangChain beépített sablonjai angolul vannak. Hiába van benne, hogy a kapott szövegek nyelvén válaszoljon, a nyelvi modell ezt figyelmen kívül hagyta.

Ennek a problémának a kiküszöbölésére magyar promptokat hoztam létre a beépítettek alapján. Ehhez két külön chaint kellett készítenem. Az első egy sima LLMChain, melynek olyan sablont adtam meg, amiben a chat előzmények és a felhasználótól kapott input szerepel. Ezenkívül a promptban szerepelnie kell a feladatnak, hogy egy önálló kérdést fogalmazzon meg az előzményekből az inputra vonatkozóan. Ezt többször kellett módosítanom a későbbiekben is, mert különböző hibák merültek fel a használat és tesztek során.

A második chain egy StuffDocumentChain, mely tartalmaz egy másik LLMChain-t, amibe a kapott dokumentumokat a megfelelő nevű változóba rakja. Itt meghatározhatjuk, hogy milyen karakterekkel válassza el az adatforrásokat, rakjon-e hozzájuk plusz szöveget. Én ezeket az alapértelmezett két üres soron és csak magán a dokumentum szövegén hagytam. Az LLMChain tartalmazza azt a promptot, amely a nyelvi modelltől a források alapján a kérdés megválaszolását kéri. Itt én megadtam ezen felül még azt is, hogy a bot egy BME VIK szakmai gyakorlatos kérdéseire válaszoló chatbot, magyarul kell válaszolnia, és mondja azt, hogy nem tud válaszolni, ha a kapott dokumentumokban nem talál releváns információkat a kérdésre.

```
qa_template = """A BME VIK szakmai gyakorlat kérdéseire válaszoló chatbot
vagy. A kérdésekre magyarul válaszolj!
Használd az alábbi dokumentumrészleteket forrásként a felhasználó kérdésének
megválaszolásához!
Ha azokból nem tudsz megadni releváns választ, akkor válaszold azt, hogy
"Sajnos erre nem tudok válaszolni, kérdezz mást a BME VIK szakmai
gyakorlattal kapcsolatban"

DOKUMENTUMRÉSZLETEK:
{context}

ÚJ INPUT: {question}

Válaszolj a kérdésre!"""
```

Ezt a notebookot lemásoltam, hogy más modellekre is megcsináljam külön a botot. Elsőként a finomhangolt huBERT modellel alakítottam ki egyet, azonban ezt nem használhattam ugyanúgy, mint az előző helyzetben az GPT-3.5-öt, hiszen ez egy kapott kontextusból válaszol a kérdésre, nem egyben kapja meg a kettőt. Ez azt is jelentette, hogy az előzményekből értelmes kérdést generáló lépést nem tudtam ezzel megoldani, így arra ugyanolyan LLMChain-t használtam, mint az OpenAI-os verzióban. Egy függvényt írtam, ami először legenerálja az új kérdést, ha nem az elején vagyunk a beszélgetésnek, majd dokumentumokra keres rá és összefűzi azokat egy stringgé, végül a modell megkapja a kérdést és a kontextust, és válaszol.

Az OpenAI-hoz hasonlóan az mGPT-vel és a PULI-GPTrioval is elkészítettem ezeket, hogy megvizsgáljam, milyen eredményeket adna egy ilyen chatbot. Sokat nem kellett változtatnom a kódon, csak egy HuggingFacePipeline-t kellett létrehoznom a megfelelő modellel és azt adni az LLMChain llm paraméterének a ChatOpenAI helyett. Ezek a modellek túl nagyok voltak a lokális futtatáshoz és az ingyenes Colab futtatásához, mert nem férnek bele 16 gigányi memóriába, így arra az elhatározásra jutottam, hogy pár hónapra előfizetek a Google Colab Pro-ra, hogy tudjak haladni a munkával ilyen irányba is.

5.5 Tesztelések

Ezután a fent vázolt különböző chatbot változatokat teszteltem le. A már korábban is használt tesztfájl mellé készítettem egy másikat is, ami kifejezetten összefüggő kérdésekre koncentrál, hogy megtudjam tényleg képesek-e az egyes chatbotok felismerni, hogy mire vonatkozik egy-egy előzményre visszautaló[38] kérdés.

Az értékelést a LangChain segítségével végeztem. Az evaluation modulon belül több kiértékelésre alkalmas chain is megtalálható, amelyek közül az QAEvalChaint használtam. Ez összehasonlítja az elvárt és a generált válaszokat egy nagy nyelvi modell segítségével. A minél jobb válaszok érdekében itt is a GPT-3.5-öt használtam.

A PULI-GPTrio és az mGPT esetében is a korábban, online tapasztalt eredményeknél is rosszabbakat kaptam. Bármilyen kérdést tettem fel, 3 karakternél többet nem akartak generálni, így ezeket a chatbotokat el is vetettem.

[38] a testing mappában a test_chat.csv fájlban találhatóak

A huBERT modelles megoldás többnyire jó válaszokat adott, azonban szembetűnő, hogy nem válaszol szép kerek mondatokban. Még, ha tudja a választ, akkor is csak konkrétan azt az 1-2 szót adja meg, ami a lényeges rész. Ez nem feltétlen baj, de az, hogy még mondatba sincs foglalva az üzenet, eléggé megtöri a természetes nyelvérzetet. Két embeddinggel is kipróbáltam: az OpenAIEmbeddinggel és az Instructorral.

Kérdés	Valós válasz	Chatbot válasz	LLM értékelés
Mennyit kell dolgozni a gyakorlat elfogadásához?	240 vagy 320 órát a képzésedtől függően.	240/320 munkaóra	CORRECT
Félév közben mennyit szabad hetente dolgozni?	20 órát engedélyezett.	maximum heti 20 óra	CORRECT
Mi van, ha nem fejezem be a szakmai gyakorlatot a nyáron?	Az őszi félévben befejezhető.	ajánlott, hogy a vizsgaidőszak utáni hét első munkanapján kezdődjön	CORRECT

OpenAIEmbeddinges eredmények részlete

Kérdés	Valós válasz	Chatbot válasz	LLM értékelés
Mennyit kell dolgozni a gyakorlat elfogadásához?	240 vagy 320 órát a képzésedtől függően.	egy héttel	CORRECT
Félév közben mennyit szabad hetente dolgozni?	20 órát engedélyezett.	őszi félévben	CORRECT
Mi van, ha nem fejezem be a szakmai gyakorlatot a nyáron?	Az őszi félévben befejezhető.	javítania kell a beszámolóját	CORRECT

Instructoros eredmények részlete

Az önálló kérdéses táblázatok részleteiből látható, hogy az OpenAI-os embeddinggel jobb eredményeket kapunk, illetve az is kiderül, hogy az értékelést végző QAEvalChain elég sokszor hibázik, nem ad megbízhatóan válaszokat. Mindkét esetben az összefüggő beszélgetést imitáló teszteknél sokkal rosszabb eredményeket kaptam, mint a csak egy kérdésre koncentrálóknál.

A GPT-3.5-ös teszteknél az eredmények mondatokba foglaltak, értelmezhetőek voltak. Ráadásul sokkal jobbak a válaszok is mind az önálló, mind az összefüggő esetben. Az értékelés hibája továbbra is fennáll pár esetben.

Kérdés	Valós válasz	Chatbot válasz	LLM értékelés
Mennyit kell dolgozni a gyakorlat elfogadásához?	240 vagy 320 órát a képzésedtől függően.	A szakmai gyakorlat elfogadásához 240 vagy 320 munkaórányi munkát kell végezni.	CORRECT
Félév közben mennyit szabad hetente dolgozni?	20 órát engedélyezett.	A szorgalmi időszakban maximum heti 20 óra gyakorlati munka végezhető.	CORRECT
Mi van, ha nem fejezem be a szakmai gyakorlatot a nyáron?	Az őszi félévben befejezhető.	Ha nem fejezed be a szakmai gyakorlatot a nyáron, akkor a következő őszi félévben kell felvenned a szakmai gyakorlat tárgyat.	CORRECT

GPT-3.5 eredmények részlete

5.5.1 ROUGE érték

A tesztek kiértékeléséhez ROUGE (Recall-Oriented Understudy for Gisting Evaluation) értéket [27] is használtam, hogy számokkal is szemléltetni tudjam az eredményeket. Eredetileg szövegösszegzések értékelésére találták ki ezt a módszert, ami az egyező n-gramok (n hosszúságú szókapcsolatok) alapján 3 értéket is megad. A recall (visszahívás) az az egyező n-gramok száma osztva az elvárt válaszban lévőkkel, a precision (pontosság) az egyezők száma a generált kimenetben lévő n-gramok számával osztva, és az f mérték, ami az előző két szám harmonikus közepe. A ROUGE-1 az unigramokra (szavakra), a ROUGE-2 a 2 hosszú szókapcsolatokra adja meg ezeket a számokat. A ROUGE-L a legnagyobb egyező szekvencia hosszából és az unigramok számából számol.

Ezeket a rouge-score[39] Python könyvtárat használva csak a huBERT és a GPT-3.5-ös tesztekre számoltam ki. Íme az eredmények:

- huBERT, OpenAI embeddinggel
    - ROUGE-2: recall: 0,143, precision: 0,163, f-érték: 0,145
    - ROUGE-L: recall: 0,212, precision: 0,297, f-érték: 0,225
- huBERT, Instructor embeddinggel
    - ROUGE-2: recall: 0,135, precision: 0,142, f-érték: 0,134
    - ROUGE-L: recall: 0,189, precision: 0,256, f-érték: 0,192
- GPT-3.5
    - ROUGE-2: recall: 0,143, precision: 0,163, f-érték: 0,145
    - ROUGE-L: recall: 0,212, precision: 0,297, f-érték: 0,225

A fenti értékekből látszik. hogy az instructoros huBERT gyengébb eredményeket ér el, mint a másik kettő, amikre teljesen ugyanazok az értékek jönnek ki. Ez abból fakad, hogy a tesztekre megfogalmazott válaszok rövidek, csak a lényegi tartalmat írtam bele. Egy szép kerek mondatokban fogalmazott tesztkészlet esetén a GPT-3.5 lenne a jobb.

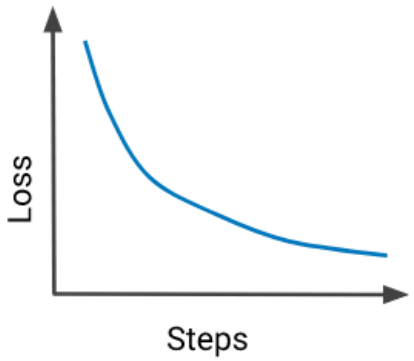
5.6 Modellek finomhangolása

5.6.1 mT5

A korábbi megoldásokon felül a mT5-öt Google Colabon belül finomhangoltam. Ehhez az adathalmaz a kérdés-válasz párokat tartalmazó CSV fájlok összessége volt. A kiinduló alap a kódhoz az elsőként elvégzett kurzus[40] finomhangolással foglalkozó laborja volt. A modell típusa nagyon hasonló volt, így lényegében csak a tanító adathalmazt tokenizáló függvényen, meg a modell nevén kellett módosítanom.

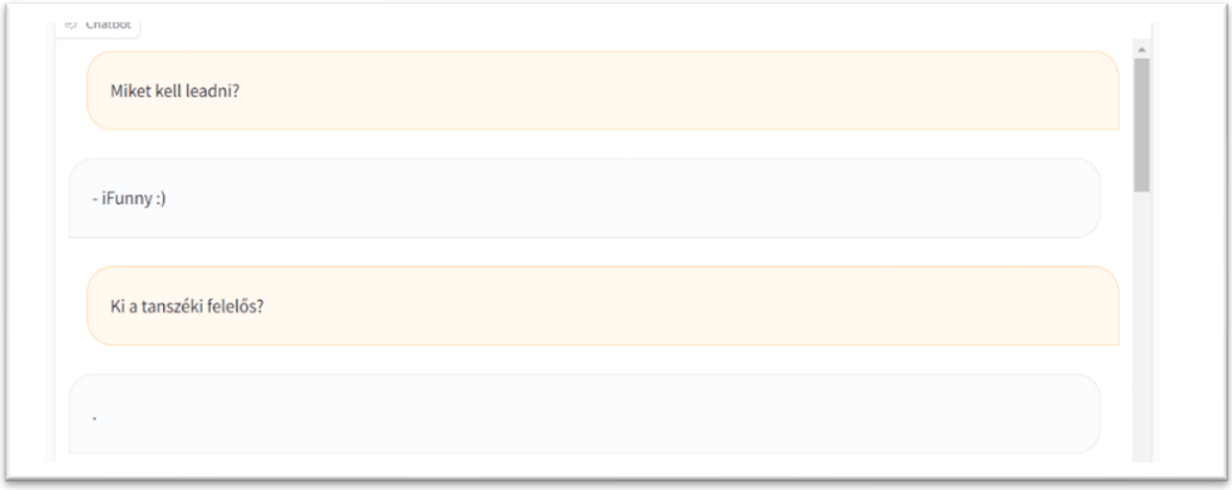
[39] https://pypi.org/project/rouge-score/

[40] https://www.coursera.org/learn/generative-ai-with-llms



5.1. ábra Ideális tanítás esetén a görbe, forrás: [28]

Nem sikerült megtalálni a megfelelő paramétereket ekkora adathalmaz esetében ahhoz, hogy az 5.1-es ábrán látható ideális training loss (tanítási veszteség) görbéhez hasonlót kapjak. Jellemzően 15 és 10 közötti értékeket sikerült kapnom, amik nem adták ki ezt a formát. A finomhangolt modell, nem tanult rá elég jól a kérdés válaszok párokra. Az 5.2-es ábrán látható, hogy teljesen értelmetlen választ adott, így el is vetettem ezt a lehetőséget.



5.2. ábra mT5 finomhangolva

5.6.2 Llama2 7b chat

A félév vége felé, mikor már a GPT-3.5-ös chatbot Azure-ben működőképes telepítésével is megvoltam, megkaptam az elérést a Meta-tól a Llama 2 modellekhez. Ezek közül a 7 milliárd paraméteres chatre finomhangolt változatot használtam. További finomhangolás nélkül a modell minden kérdésre azt mondta, hogy nem tud válaszolni, majd bemásolta az összes megtalált dokumentumot (5.3 ábra).

5.3. ábra Llama2 chat egyik eredménye

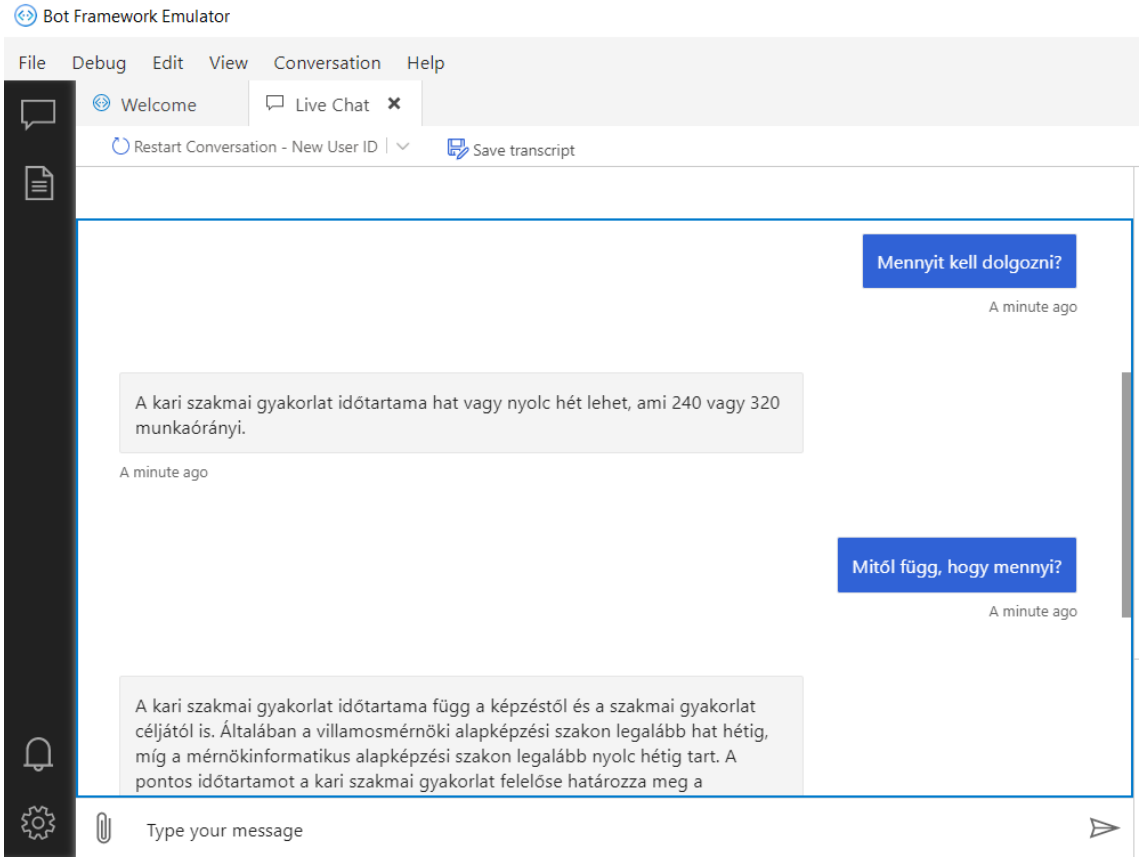
A finomhangoláshoz ugyanazt az adathalmazt próbáltam használni, amit az előző esetben. Több alkalommal is CUDA Out-of-memory hibát kaptam, ezért megpróbáltam optimalizálni a tanítást 1 GPU-ra [29]. Ezzel el tudtam érni, hogy ne használjon túl sok memóriát, de ekkor batch méret nem megfelelő hibát kaptam. Ezt a tokenizáló függvény módosításával tudtam volna elérni, azonban időszűke miatt már nem tuftam ezzel foglalkozni.

5.7 Azure telepítés és Teams elérés

A tesztek eredményei alapján a GPT-3.5-öt használó chatbottal valósítottam meg a Microsoft Teams integrációt. A korábbi botot a Microsoft Bot Framework-öt használva alakítottam át ahhoz, hogy Azure-be lehessen telepíteni, majd el lehessen érni egyszerűen Teams-ből chaten keresztül.

A kód kiindulópontja az Echobot nevű üzeneteket visszaküldő mintaprogram[41] volt. A kódban lévő Bot osztályt módosítottam úgy, hogy az inicializálásnál elvégezze az az adatbázis betöltését vagy annak nem létezése esetén annak felépítését, valamint a ConversationalRetrievalChaint is hozza létre. Az on_message_activity() függvényt pedig úgy definiáltam felül, hogy az a generált választ küldje vissza a felhasználónak.

[41] Bevezető a használathoz: https://learn.microsoft.com/en-us/azure/bot-service/bot-service-quickstart-create-bot?view=azure-bot-service-4.0&tabs=python%2Cvs



5.4. ábra Csevegés az emulátorból

Az app.py-t futtatva elindítottam a szervert lokálisan, és a Bot Framework Emulatorral rácsatlakoztam (5.4 ábra). Pár üzenetet elküldve megbizonyosodtam a működés helyességéről. Próbáltam több párhuzamos chaten is, ekkor szembesültem egy újabb problémával. A chatbotnak 1 darab közös memóriája volt, ami azt jelentette, hogy az összes felhasználó üzenetét abban tárolta el, nem pedig külön-külön. Adatvédelmi szempontból nincs olyan megoldás a keretrendszerben, ami egy-egy felhasználó előzményeit tárolja, így valami más megoldást kellett találnom.

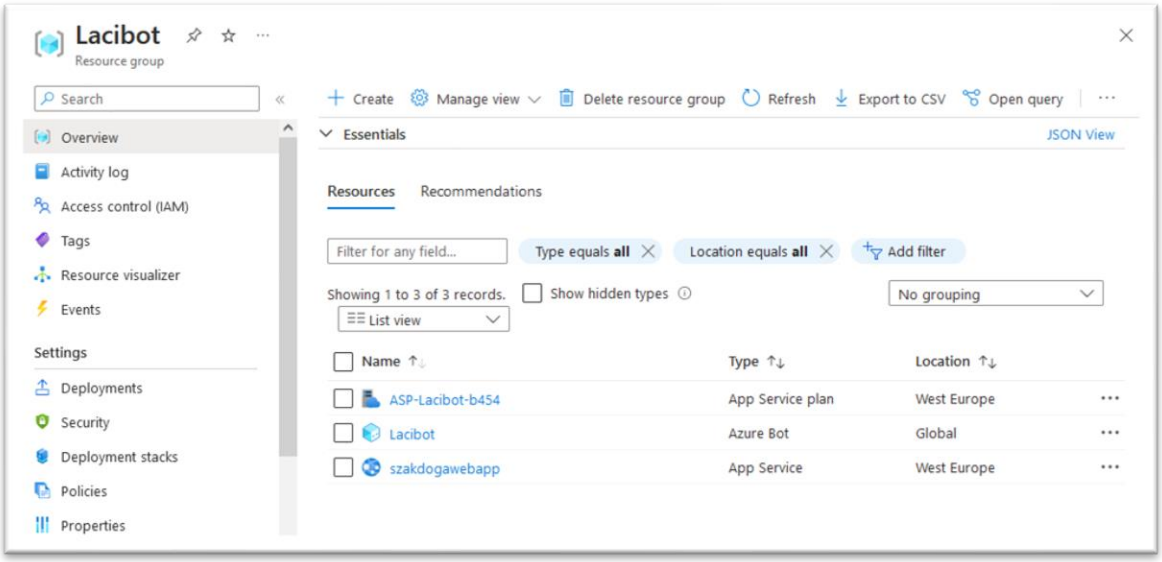
Az állapot tárolására is létezik több microsoftos példakód, ezek közül a state-managementből[42] indultam ki. Nekem csak a ConversationState-re volt szükségem ebben az esetben, a UserState-re nem. A különböző chat előzménynek tárolására azt a megoldást találtam ki, hogy minden beszélgetésnek lesz egy saját memóriája. Ehhez létrehoztam egy ConversationHistory osztályt, amiben csak egy BaseChatMemory típusú attribútum van. Az on_message_activity()-t módosítani kellett úgy, hogy az elején lekéri a saját

42 https://github.com/microsoft/BotBuilder-Samples/tree/main/samples/python/45.state-management

ConversionState tárolójából azt a ConversationHistory típusú objektumot, ami az adott eseményhez tartozó TurnContexthez tartozik. Ha ez az elem nem létezik, akkor létrehozok egy új memóriát. A válasz generálása előtt a chainben kicserélem a memóriát az aktuálisra, hogy azok alapján fusson le. A memória változásait elmentem a minden eseményre lefutó on_turn() függvényben. on_message_activity():

```
async def on_message_activity(self, turn_context: TurnContext):
     try:
          conversation_data = await self.conversation_history_accessor.get(
               turn_context, ConversationHistory
          )
          if conversation_data.memory is None:
               conversation_data.memory = ConversationBufferWindowMemory(
                    k = self.memory_k,
                    memory_key="chat_history",
                    return_messages=True
               )
          answer = await self.get_answer(
               turn_context.activity.text, conversation_data.memory
          )
          await turn_context.send_activity(answer)
     except:
          await turn_context.send_activity("Hiba lépett fel, kérlek próbáld
újra egy kis idő múlva!")
```

A fenti probléma megoldása után a következő lépés az Azure-publikálása volt. Ehhez először is regisztrálnom kellett az Azure-portálon, majd létrehoztam egy erőforrás-csoportot. Régen elérhető volt egy Azure Bot Service nevű szolgáltatás, amely mindent létrehozott, amire szükségem lett volna, de már egyesével kellett ezt megtennem. A csoporton belül először egy Web Appot hoztam létre Linux operációs rendszerrel, Python 3.11-es környezettel a nyugat-európai régióban. Ehhez egy Service plan (szolgáltatás milyenségét meghatározó erőforrás) is létrejött. Ennél először az ingyenes, napi 60 perc futtatást lehetővé tévőt választottam, de a későbbi, mások általi tesztelhetőség érdekében a legolcsóbb folyton futóra váltottam. A másik elem, amit hozzá kellett adnom a csoporthoz, egy Azure Bot volt. Ennél az volt fontos, hogy Multi Tenant legyen beállítva az app típusnál, mert a dokumentáció szerint a Python botok csak ilyenek lehetnek. Az elkészülte után a konfigurációban a webappunk API endpointját kell megadni, a csatornák menüben pedig egy Microsoft Teams-csatornát kell létrehozni. Az erőforráscsoport ekkor az 5.5-ös ábrán látható módon épült fel.

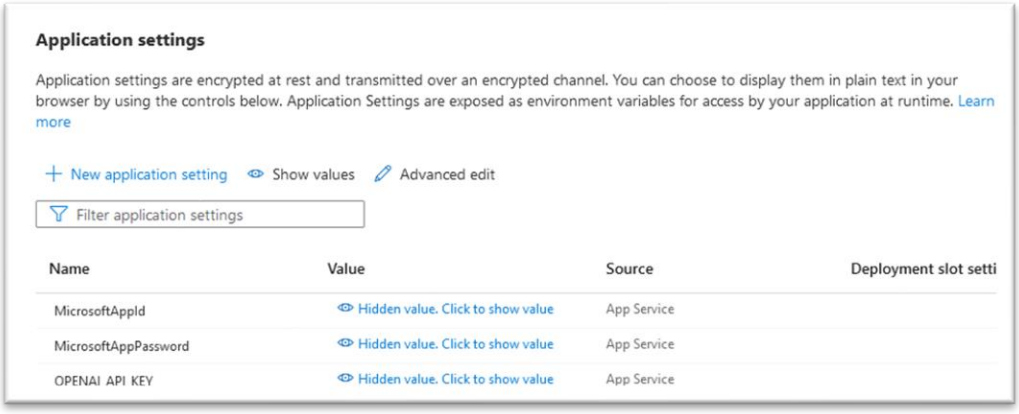


5.5. ábra Lacibot resource group

Ezt követően a kód feltöltéséhez a VS Code-ban az Azure bővítményen belül kiválasztottam a szakdogawebapp nevű App Service-t és a Deploy to Web App menüpontnál pedig a Bot Framework-ös mappát. Első alkalommal hibaüzenetet kaptam, ugyanis a felhőben lévő Python könyvtárak kicsit elavultak, így a Chroma jelezte a hibát, hogy az általa használt pysqlite3-nak egy túl régi verziója van fent. Szerencsére a hibaüzenet tartalmazott egy linket[43] is a megoldáshoz. A probléma megoldása után már futott a botom a felhőben, mégse tudtam vele kommunikálni. Hosszas kutatás után kiderült, hogy a dokumentumokban pár helyen opcionálisnak írt dolog Azureben kötelező, anélkül nem fut a program[44]. MicrosoftAppId-t és MicrosoftAppPaswword-öt kellett létrehoznom, és környezeti változóként megadnom a felhőben, azonban az oldal, ahol ezt meg tudtam volna tenni, zárolva volt előlem, mert intézményi szinten nincs jogom hozzá. Szerencsére van egy VS Code bővítmény, a saját id-jainkat és hozzá tartozó jelszavainkat kezelni, így ott be tudtam állítani ezeket, majd pedig Azure-ben használni (5.6 ábra).

[43] https://docs.trychroma.com/troubleshooting#sqlite

[44] https://stackoverflow.com/questions/74177322/bot-framework-emulator-works-but-web-chat-not-does-not



5.6. ábra Környezeti változók a szakdogawebappnál

A webes chaten ezek után már tudtam kommunikálni a bottal, de Teams-en keresztül még mindig nem. Ennek az volt az oka, hogy intézményi szinten le van tiltva az ilyen appok hozzáadása. Írtam egy emailt az egyetemi supportra azzal a kéréssel, hogy engedélyezzék az appot számomra, melynek hatására kaptam egy egyedi policy-t, aminek köszönhetően tudok üzenetet küldeni a botnak (5.7 ábra).

5.7. ábra Lacibot Teams-en

Később kiderült, hogy mobilról megnyitva a linket, BME-s fiókkal az applikációból tud más is üzenetet küldeni neki. A mások általi tesztelések alapján javítottam a prompt megfogalmazásán, például sokszor témaváltásnál mindenáron az előző kérdést is bele akarta fűzni az új kérdésbe. A javításokat mindig kitelepítettem az Azure-be is.

6 Összegzés

6.1 Az eredmények értékelése

A félév során az elvégzett munka alatt megismertem a nagy nyelvi modellek belső működését és a különböző, tudásukat bővítő lehetőségeket. Ez a terület rendkívül fontos lesz az elkövetkezendő években, hiszen számos helyen egyszerűsíthetnek nagy nyelvi modellekkel különböző munkákat. Egy ügyfélszolgálatba lehet chatbotokat építeni, vállalatok a saját szabályzatukból vagy akár egy általuk fejlesztett program dokumentációjából is készíthetnek ilyen jellegű botokat, hogy segítség azok a céghez frissen odakerült munkatársakat.

A megismert tudást felhasználva elkészítettem a szakmai gyakorlat chatbot prototípusát, amelyet ma már Microsoft Teams-en keresztül is el lehet érni. Ehhez egyedi policy-beállításokat is kellett kérnem az egyetemi Teams-üzemeltetéstől.[45] A bot minden felhasználónak a saját beszélgetéséhez tartozó kontextust figyelembe véve tud válaszolni. Ez azt jelenti, hogy figyelembe veszi a korábbi üzenetváltást, hogy tudja, mire vonatkozik egy újabb kérdés.

A bot elkészítéséhez többféle megoldást is megvizsgáltam a félév során, melyek közt több eltérő működési elvű modell is megtalálható. Ezek közül a leghatékonyabbat, az OpenAI API-t használót választottam ki. Az ezen keresztül elérhető modellek közül a GPT-3.5-höz intéz hívást a chatbot. A hatékonyabb működés érdekében ezt GPT-4-re egyszerűen ki lehet cserélni, de ez a költségek jelentős növekedésével is jár.

Nyitott kérdés a bot üzemeltetése. Például, hogy hogyan lenne legegyszerűbben intézményi szinten is elérhetővé tenni a hallgatók számára Teamsen. Ha tényleg alkalmazni szeretné a tanszék, akkor milyen formában és mennyit szeretne költeni arra, hogy fusson egy ilyen bot.

6.2 Továbbfejlesztési lehetőségek

Az egyik továbbfejlesztési lehetőség a chatbot számára, hogy egy REST API-n keresztül elérje az aktuálisan kiírt témákat, így a segítségével a felhasználók akár

[45] Videó pár üzenetről: https://tinyurl.com/3adns65t

gyorsabban kereshetnek nekik megfelelő munkát. Ebben az esetben még azt is meg tudná mondani, hogy van-e még szabad hely, mennyi a fizetés, milyen technológiák ismerete szükséges stb. Hasonló módon a szerződött cégek listáját is valós időben lehetne lekérdezni.

Az Azure Bot segítségével akár egy email címhez is lehet kötni egy chatbotot. Számos hallgató jobban hozzá van szokva, hogy ezen keresztül tegye fel a szakmai gyakorlatos kérdését. Emiatt célszerű lenne bővíteni a botot egy olyan résszel, amikor nem chaten keresztül, hanem emailben is válaszolni tudna. A bot működését is érdemes időnként felülvizsgálni, mert e gyorsan fejlődő kutatási területen gyakran jelennek meg más, esetleg pontosabb válaszokat adni képes vagy egyszerűbb megoldások.

Egy ilyen chatbot mintájára a későbbiekben több hasonló is készülhet. Nagyon hasznos lehet egy az egyetemre érkező elsőéves hallgatók számára elérhető chatbot, amely segít nekik az egyetem szabályzatában eligazodni, esetleg felsőbb éves hallgatóktól származó tippeket is tud adni nekik különböző, az egyetemi tanulmányokkal és élettel kapcsolatos témákban.

Köszönetnyilvánítás

Szeretnék köszönetet mondani konzulensemnek, Dr. Forstner Bertalannak a félévben kapott segítségért, javaslatokért, mind a chatbot elkészítéséhez, mind a dolgozatíráshoz kapott tanácsokért. Rajta kívül szeretném még megköszönni Dr. Blázovics Lászlónak a segítséget az adatbázis bővítéséhez.

Irodalomjegyzék

[1] Exploding Topics, Fabio Duarte: *Number of ChatGPT Users (Nov 2023)*, https://explodingtopics.com/blog/chatgpt-users. (2023. 11. 13.)

[2] Elastic: *What is a large language model (LLM)?*, https://www.elastic.co/what-is/large-language-models. (2023. 11. 19.)

[3] IBM: *What are recurrent neural networks?*, https://www.ibm.com/topics/recurrent-neural-networks . (2023. 11. 19.)

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: *Attention Is All You Need*, arXiv:1706.03762v7 (2023. 11. 20.)

[5] Medium: *Transformer Architecture explained*, https://medium.com/@amanatulla1606/transformer-architecture-explained-2c49e2257b4c (2023. 11. 20.)

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.04805v2 (2023. 11. 21.)

[7] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever: *Improving Language Understanding by Generative Pre-Training*, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (2023. 11. 21.)

[8] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer: *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*, arXiv:1910.13461v1 (2023. 11. 21.)

[9] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, Laurent Sifre: *Training Compute-Optimal Large Language Models*, arXiv:2203.15556v1 (2023. 11. 22.)

[10] Google: *Introducing FLAN: More generalizable Language Models with Instruction Fine-Tuning*, https://blog.research.google/2021/10/introducing-flan-more-generalizable.html (2023. 11. 22.)

[11] Vladislav Lialin, Vijeta Deshpande, Anna Rumshisky: *Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning*, arXiv:2303.15647v1 (2023. 11. 22.)

[12] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen:
LoRA: Low-Rank Adaptation of Large Language Models, arXiv:2106.09685v2 (2023. 11. 23.)

[13] Meta: *Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models*, https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/ (2023. 11. 22.)

[14] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, Ilya Sutskever: *Robust Speech Recognition via Large-Scale Weak Supervision*, arXiv:2212.04356v1 (2023. 11. 27.)

[15] Keivalya Pandya, Mehfuza Holia: *Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations*, arXiv:2310.05421v1 (2023. 11. 25.)

[16] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, Samuel Weinbach:
GPT-NeoX-20B: An Open-Source Autoregressive Language Model, arXiv:2204.06745v1 (2023. 11. 26.)

[17] Zijian Győző Yang, Réka Dodé, Gergő Ferenczi, Enikő Héja, Kinga Jelencsik-Mátyus, Ádám Kőrös, László János Laki, Noémi Ligeti-Nagy, Noémi Vadász, Tamás Váradi:
Jönnek a nagyok! BERT-Large, GPT-2 és GPT-3 nyelvmodellek magyar nyelvre, https://rgai.inf.u-szeged.hu/sites/rgai.inf.u-szeged.hu/files/mszny2023_0.pdf (2023. 11. 26.) (XIX. Magyar Számítógépes Nyelvészeti Konferencia 247. oldal)

[18] Zijian Győző Yang, László János Laki, Tamás Váradi, Prószéky Gábor:
Mono- and multilingual GPT-3 models for Hungarian, http://real.mtak.hu/173960/1/TSD_2023_GPT.pdf (2023. 11. 26.)

[19] Nemeskey Dávid Márk: *Introducing huBERT*, https://hlt.bme.hu/media/pdf/huBERT.pdf (2023. 11. 26.)

[20] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, Colin Raffel:
mT5: A massively multilingual pre-trained text-to-text transformer, arXiv:2010.11934v3 (2023. 11. 27.)

[21] Meta: *Llama 2: Open Foundation and Fine-Tuned Chat Models*, https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/ (2023. 11. 27.)

[22] Oleh Shliazhko, Alena Fenogenova, Maria Tikhonova, Vladislav Mikhailov, Anastasia Kozlova, Tatiana Shavrina: *mGPT: Few-Shot Learners Go Multilingual* arXiv:2204.07580v2 (2023. 11. 27.)

[23] OpenAI: *Introducing ChatGPT and Whisper APIs*, https://openai.com/blog/introducing-chatgpt-and-whisper-apis (2023. 11. 27.)

[24] OpenAI: *GPT-4*, https://openai.com/research/gpt-4 (2023. 11. 27.)

[25] Nils Reimers, Iryna Gurevych: *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*, arXiv:1908.10084v1 (2023. 11. 30.)

[26] Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A. Smith, Luke Zettlemoyer, Tao Yu: *One Embedder, Any Task: Instruction-Finetuned Text Embeddings*, arXiv:2212.09741v3 (2023. 11. 30.)

[27] Chin-Yew Lin: *ROUGE: A Package for Automatic Evaluation of Summaries*, https://aclanthology.org/W04-1013.pdf (2023. 12. 03.)

[28] Googel: *Interpreting Loss Curves*, https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic (2023. 12. 04.)

[29] Hugging Face: *Methods and tools for efficient training on a single GPU*, https://huggingface.co/docs/transformers/perf_train_gpu_one (2023. 12. 04.)