



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

MESTERSÉGES INTELLIGENCIA

TANSZÉK

Digitális Asszisztens Felvételizőknek

Témavezető:

Gyöngyössy Natabara

Phd hallgató

Szerző:

Tóth András Tamás

programtervező informatikus BSc

Budapest, 2024

Tartalomjegyzék

1. Bevezetés	4
2. Specifikáció	5
2.1. Program leírása	5
2.1.1. Felhasználói felület	5
2.1.2. Kiszolgáló szerver	5
2.1.3. Mesterséges intelligencia modell és vektor adatbázis	6
2.1.4. Pontszámító eszköz	6
2.1.5. Karbantartó program	6
2.2. Felhasználói csoportok	6
2.3. Megszorítások	7
2.3.1. Kiszolgáló kapacitás	7
2.3.2. Finomhangolás	7
2.3.3. Pontosság	7
2.3.4. Értékelés	7
2.4. Függőségek	8
2.4.1. Nyelvi modell	8
2.4.2. Futtatási környezet	8
2.5. Követelmények	8
2.5.1. Funkcionális követelmények	9
2.5.2. Nem funkcionális követelmények	10
2.6. Használati esetek	10
2.6.1. Felvételiző használati esetei	10
2.6.2. Karbantartó használati esetei	11
3. Fejlesztői dokumentáció	12
3.1. Bevezetés	13
3.2. Nyelvi modellek	13

3.3.	Felhasználói felület	14
3.4.	Feldolgozó egység	16
3.4.1.	Chat motor	17
3.4.2.	Kereső motor	19
3.4.3.	Válasz formázása	22
3.4.4.	Pontok kiszámolása	23
3.4.5.	Promptok	24
3.4.6.	Speciális eljárások	25
3.5.	Karbantartó alkalmazások	26
3.5.1.	Adatbázis kezelő	26
3.5.2.	Modell kiértékelő	27
3.6.	Konténerizáció és futtatás	27
3.7.	Egységtesztek	28
3.8.	Ismert hibák	29
3.9.	Osztály szintű dokumeentáció	29
4.	Felhasználói dokumentáció	30
4.1.	Karbantartó	30
4.1.1.	Áttekintés	30
4.1.2.	Előkészületek	30
4.1.3.	Programcsomag futtatása	30
4.1.4.	Adatbázis kezelése	31
4.1.5.	Mesterséges intelligencia kiértékelése	32
4.1.6.	Főprogram konfigurálása	33
4.1.7.	Program frissítése	34
4.1.8.	Futtatási követelmények	35
4.2.	Felhasználó	36
4.2.1.	Használat	36
4.2.2.	Használati követelmények	37
	Irodalomjegyzék	39
A.	Osztály szintű dokumentáció	41
A.1.	Namespace List	41
A.2.	Class List	41

A.3. chat_engines Namespace Reference	42
A.3.1. Detailed Description	42
A.4. configs Namespace Reference	42
A.4.1. Detailed Description	42
A.5. tools Namespace Reference	43
A.5.1. Detailed Description	43
A.6. utils Namespace Reference	43
A.6.1. Detailed Description	43
A.7. utils.chat_history.ChatHistory Class Reference	43
A.7.1. Detailed Description	43
A.7.2. Constructor & Destructor Documentation	44
A.7.3. Member Function Documentation	44
A.8. chat_engines.controller_chat_engine. ControllerChatEngine Class Reference	45
A.8.1. Detailed Description	46
A.8.2. Constructor & Destructor Documentation	47
A.8.3. Member Function Documentation	47
A.9. gui.GradioGUI Class Reference	49
A.9.1. Detailed Description	49
A.9.2. Constructor & Destructor Documentation	50
A.9.3. Member Function Documentation	50
A.10.configs.pont_ik_config.PontIkConfig Class Reference	52
A.10.1. Detailed Description	52
A.11.utils.question_former.QuestionFormer Class Reference	52
A.11.1. Detailed Description	53
A.11.2. Constructor & Destructor Documentation	53
A.11.3. Member Function Documentation	54
A.12.tools.rerank_query_engine RerankQueryEngine Class Reference . . .	55
A.12.1. Detailed Description	56
A.12.2. Constructor & Destructor Documentation	56
A.12.3. Member Function Documentation	56
A.13.eval_model.TestChatEngine Class Reference	58
A.13.1. Detailed Description	58
A.13.2. Member Function Documentation	58

1. fejezet

Bevezetés

A 2024-es évtől a felsőoktatási felvételi eljárások nagyban módosultak, és a felvételi pontok követelményeinek egy része az intézményekre lett bízva. Ez intézményenként és szakonként eltérő követelményeket eredményezhet, ami nehézséget okozhat egy felvételizőnek.

A folyamat megkönnyítése érdekében készül ez a program, ami lehetővé teszi a szabályozással kapcsolatosan felmerülő kérdések gyors megválaszolását, valamint a várható felvételi pontok kiszámolását a felhasználó releváns adatainak felhasználásával.

A program egy nagy nyelvi modell alapú chat alkalmazás, ami az Eötvös Loránd Tudományegyetem (ELTE) Informatika Karának (IK) Program Tervező Informatikus (PTI) alapképzésére felvételizni kívánó személyeknek segít a felvételi pontok kiszámításával, illetve a felmerülő kérdések megválaszolásával. Ezt releváns dokumentumok keresésével, valamint egy pontszámító eszköz segítségével teszi, a pontosság növelése érdekében.

A program csomag és a dokumentumok karbantartásához nincs szükség a mesterséges intelligenciai eszközök ismeretére.

2. fejezet

Specifikáció

2.1. Program leírása

A program használata során egy felhasználó tud kérdéseket föltenni, majd ezt egy nagy nyelvi model, egy előre megadott dokumentum halmaz alapján, valamint egy pontszámító eszköz segítségével válaszolja meg. A nyelvi modell a beszélgetés során figyelembe veszi a korábbi üzeneteket is kontextusként.

A program öt egységből áll, és a működés során ezek közül három folyamatosan kommunikál egymással.

2.1.1. Felhasználói felület

A felhasználói felület, egy **egyszerű chat felület**, ahol a felhasználó üzenetet tud begépelni egy szöveges mezőbe, és ezt el tudja küldeni a nagy nyelvi modellnek egy gombnyomással. A szöveges mező alatt szerepel egy törlés, egy visszavonás és egy újraküldés gomb is.

A kérdésre érkező válasz a szöveges mező feletti konténerben látható, valamint a dokumentumban a modell által figyelembe vett releváns részletek a felület jobb oldalán lévő keretben található.

2.1.2. Kiszolgáló szerver

Ez az egység felel a **felhasználói munkamenetért**, valamint a felhasználói felületről érkező kérések kiszolgálásáért. A kiszolgáló szerver **kommunikál mesterséges intelligencia modellel**.

2.1.3. Mesterséges intelligencia modell és vektor adatbázis

Ez az egység a kiszolgáló szervertől kapott kérdés alapján a vektor adatbázisból szemantikai és kulcsszavas hibrid hasonlóság alapján **kiszűri a releváns dokumentumokat**, majd ezeket a kérdéssel együtt átadja a nagy nyelvi modellnek, ami ezek alapján **egy választ generál**. A választ visszaküldi a kiszolgáló szervernek, ami majd továbbítja azt.

2.1.4. Pontszámító eszköz

A felvételiző várható pontszámát a pontosság érdekében nem a modell, hanem **egy determinisztikus eszköz** végzi. A modell megadja a szükséges adatokat, majd az eszköz visszaadja a várható pontszámot, amit a modell beépít a válaszába.

A pontszámításnál figyelembe vett paraméterek:

- Kötelező érettségik szintje és eredménye.
- Idegen nyelvi érettségi szintje és eredménye.
- Választott érettségi szintje és eredménye.
- Szakképzések eredménye ha van ilyen.
- Önkéntes katonai szolgálat.
- Intézményi pontok.

2.1.5. Karbantartó program

Ez az egység egy **CLI alkalmazás**, ami lehetővé teszi a vektor adatbázisban szereplő dokumentumok törlését, és új dokumentumok létrehozását.

2.2. Felhasználói csoportok

A felhasználók két csoportba sorolhatók: ezek a felvételizők, valamint a karbantartó fejlesztők.

A **felvételizők** a felhasználói felületen keresztül kommunikálnak a nagy nyelvi modellel. Tudnak kérdést feltenni, arra választ kapni, valamint meg tudják kérni a modellt, hogy számolja ki a várható pontszámukat a megadott adatok alapján.

A **fejlesztők** a CLI alkalmazáson keresztül tudják aktualizálni a vektor adatbázisban lévő dokumentumokat. A karbantartáshoz nincs szükség a mesterséges intelligenciai eszközök ismeretére, a CLI eszköz a dokumentáció alapján előzetes tudás nélkül használható.

2.3. Megszorítások

2.3.1. Kiszolgáló kapacitás

A program **egyszerre 10-20 felhasználót** tud kiszolgálni. Ennek következtében az adatbázis skálázhatóságára nem kell különös hangsúlyt fektetni. Mivel a felhasználók csak az ELTE IK PTI BSc-re felvételizők egy csoportja ezért feltehető, hogy nincs szükség nagyobb kapacitásra.

2.3.2. Finomhangolás

A mesterséges intelligencia modell **finomhangolására nem kerül sor**. Egy nagy nyelvi modell finomhangolása kifejezetten erőforrásigényes, ezért költséges lenne ezt megtenni. A modern modellek szerencsére rendelkeznek "few-shot learning" képességekkel, így finomhangolás nélkül is elérhető a megfelelő működés.

2.3.3. Pontosság

A modell által adott válasz információ tartalma **nem garantált, hogy teljes vagy helyes**. A cél a lehető legpontosabb válaszadás a szabályozással kapcsolatos dokumentumok, illetve egy pontszám kiszámítására alkalmas eszköz használatával. **A dokumentum helyességét feltételezzük**, nem ellenőrizzük.

2.3.4. Értékelés

A válasz értékelése egy kis méretű kézi szakértői kérdés-válasz adatsoron fog zajlani. Ez a LlamaIndex [1] *FaithfulnessEvaluator* eszközével fog történni, ami egy

nagy nyelvi modellel ellenőrzi a válasz pontosságát a talált dokumentumokhoz képest. Az értékelés során a feltöltött dokumentumokat helyesnek tekintjük.

2.4. Függőségek

2.4.1. Nyelvi modell

A program Python 3.11 [2]-ben van megvalósítva. A nyelvi modellt az OpenAI api [3]-n keresztül fogja a program elérni, vagy saját modell esetén a Transformers [4] könyvtár kezeli, ami Pytorchra [5] épül. A vektor adatbázisban való keresés és a modell együttműködését a LlamaIndex [1] végzi. A használt adatbázis a Weaviate [6]. A chat felületet a Gradio [7] biztosítja.

2.4.2. Futtatási környezet

A program cloud környezetben Azure virtuális gépen fog futni, erre kell felkészíteni. A csomag futtatásának könnyítése érdekében konténerizálva lesz egy Docker image formájában. A deployment nem képzi a szakdolgozat részét, azonban a Docker telepítési és indítási útmutatóját el kell készíteni, a képfájlt elérhetővé kell tenni.

Az OpenAI API [3] kulcsot nem fordítjuk a Dockerbe, hanem kívülről megadott környezeti változóként kezeljük a végső szoftver verzióban.

A vektoradatbázis tartalma nem veszik el a virtuális gép leállításával ezért ezt egy perzisztens volume-ban tároljuk.

2.5. Követelmények

Az követelmények három szintbe sorolhatóak, ezeket a 2.1-es ábra részletezi. Az követelmények szintjét a továbbiakban az elvárás után zárójelek között lévő szám jelzi.

1. szint	Feltétlenül szükséges megvalósulnia.
2. szint	Fontos lenne, hogy megvalósuljon, de a program szállítható nélküle is.
3. szint	Nem fontos, hogy megvalósuljon, de a program értékét növelné.

2.1. táblázat. Az követelmények szintjeinek jelentése.

2.5.1. Funkcionális követelmények

Felhasználói felület

- A felhasználónak bejelentkezés nélkül tudnia kell használni a programot (1), viszont a munkamenetet kezelni kell (1).
- Több kérdésből álló beszélgetést is lehetséges folytatni (1).
- A beszélgetést el kell menteni a felhasználói oldalon a későbbi folytatáshoz (2).
- A modell válaszát meg kell jeleníteni (1).
- A dokumentumokból kinyert idézeteket meg kell jeleníteni (2).
- A dokumentumokból kinyert idézeteket a dokumentumban kijelölve kell megjeleníteni (3).

Nyelvi modell

- A nagy nyelvi modellt az OpenAI api-ja biztosítja (1).
- Saját nyelvi modell fut egy Azure virtuális gépen (3).
- A nyelvi modell működése ki van egészítve vektor adatbázis béli kereséssel (1).
- A keresés ki van egészítve a találatok újra rangsorolásával (2).
- A pontok kiszámításának pontossága érdekében ezt nem a modell végzi, hanem egy determinisztikus eszköz amit REST api-n keresztül ér el a modell szükség esetén (1).
- A feltöltött dokumentumoknak a virtuális gép leállítása után sem szabad elveszniük ezért perzistens Docker volumeon kell tárolni őket (1).

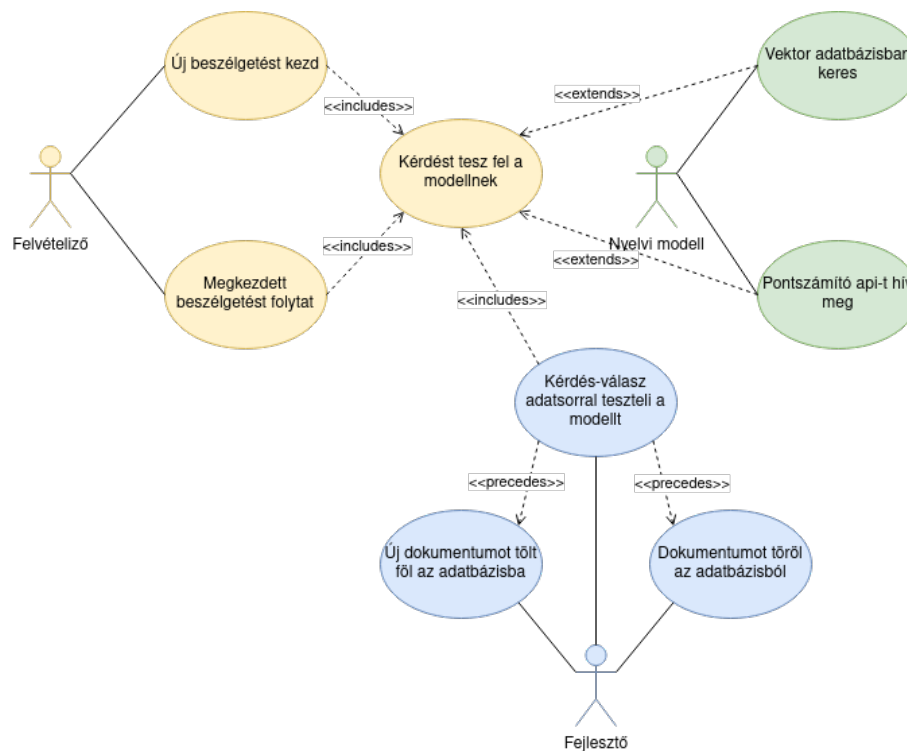
Karbantartás

- A program helyes működéséhez szükségesek az esetleges szabályváltozásokkal járó frissítések, ahol egy karbantartó fejlesztőnek tudnia kell új dokumentumot feltöltenie, valamint régit törölnie. Ezt egy CLI eszközön keresztül tudja megtenni (1).

2.5.2. Nem funkcionális követelmények

- Szükséges a rendszert tesztelni: a determinisztikus részeit egység tesztek formában, a nyelvi modell válaszait pedig egy szakértői kérdés-válasz adatsoron (1).
- A dokumentumok frissítéséhez biztosítani kell egy teszt adatbázist, ahol tesztölges kérdés-válasz adatsoron lehet tesztelni a modellt. A teszt eredménye egy mérőszám, ami könnyen értelmezhető mesterséges intelligenciával kapcsolatos ismeretek nélkül is (2).
- A felhasználónak nem kell kritikus adatokat küldenie a szervernek, viszont a https protokollal való elérés így is fontos (3).

2.6. Használati esetek



2.1. ábra. Felhasználói eset diagramm.

2.6.1. Felvételiző használati esetei

A felvételiző tud **új beszélgetést kezdeményezni**, ekkor minden eddigi üzenet törölve lesz a kontextusból, **csak a felhasználó kérdése lesz elküldve** a nyelvi

modellnek. A kérdés alapján a vektor adatbázisban a releváns dokumentumok kinyerése után ezek tartalma is hozzáadódik a kontextushoz. A nyelvi modell a kontextus alapján szükség esetén meghívja a pontszámító api-t és választ generál. A válasz visszaküldésre kerül, majd megjelenik a felhasználói felületen.

A felvételiző **már megkezdett beszélgetést is tud folytatni**, ekkor a kontextusba az aktuális kérdésén kívül a **korábbi kérdései és az ezekre kapott válaszok is bekerülnek**, és a modell ez alapján hajtja végre a korábban részletezett folyamatot.

2.6.2. Karbantartó használati esetei

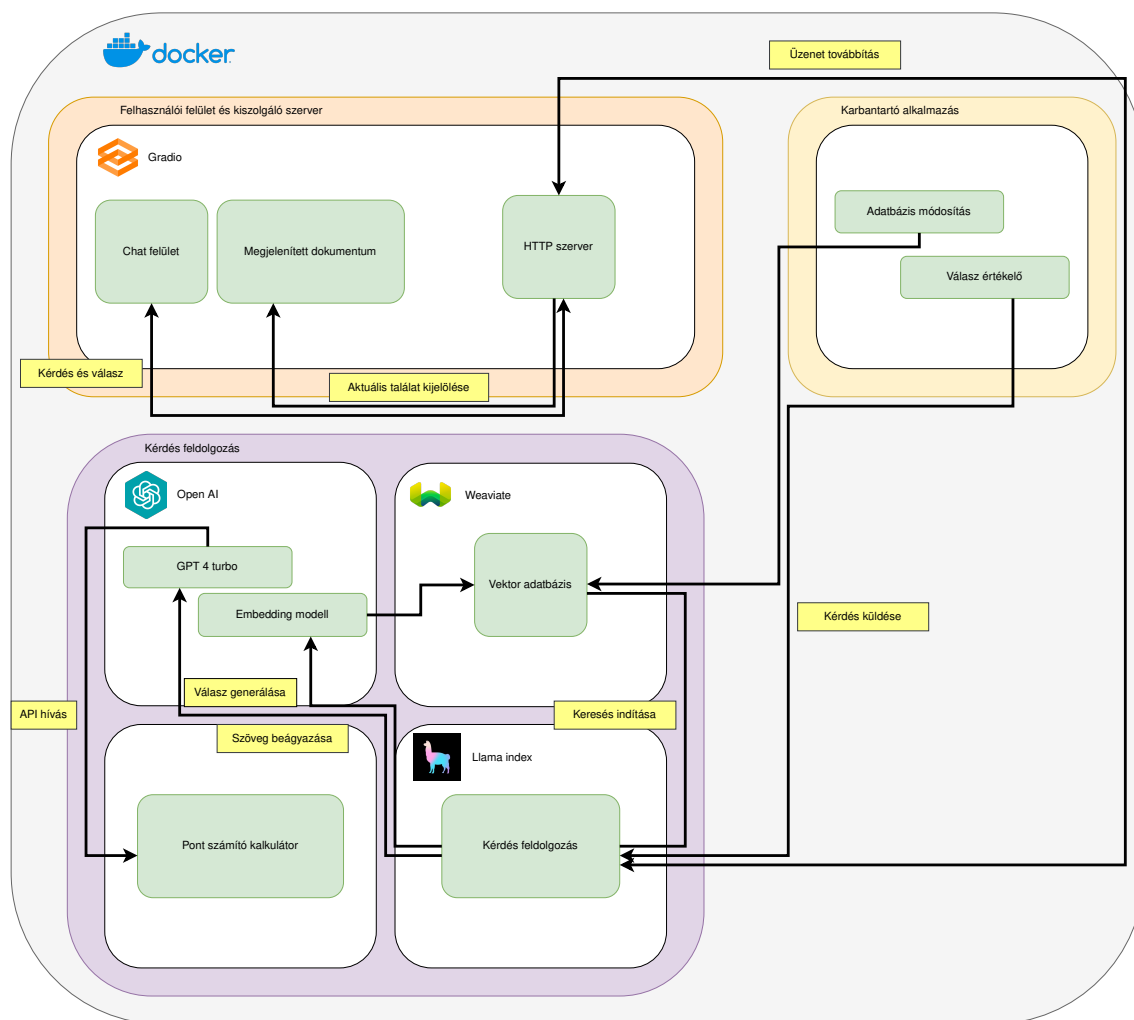
A karbantartó egy **CLI alkalmazáson** keresztül tud új dokumentumokat létrehozni, valamint meglévőket törölni az adatbázisból. Törlés esetén az **adatbázis kiürítésre kerül**, új dokumentumokat kell föltölteni. Dokumentum felvétele esetén azt földaraboljuk, valamint elmentjük.

A karbantartó tudja **tesztelni a modell működését**, erre a **dokumentumok módosításakor lehet szükség**. A teszteléshez egy kérdés-válasz adatsort kell használnia, ami alapján a modellt ki lehet értékelni. A teszt során a karbantartó egy mérőszámot kap, ami alapján el tudja dönteni, hogy megfelelően működik-e az alkalmazás.

A tesztelés során a modellnek történő kérdés megválaszolása megegyezik a felhasználó által küldött kérdés megválaszolásának folyamatával.

3. fejezet

Fejlesztői dokumentáció



3.1. ábra. Architektúra diagramm, ahol látható a program három nagy egysége: a felhasználói felület, a kérdés feldolgozó egység, valamint a karbantartó alkalmazások. A kérdésfeldolgozó egységben van a program fő logikája, hiszen ez felel a felhasználóval való beszélgetés lebonyolításáért, releváns információk kereséséért, valamint a felvételi pontszám kiszámításáért.

3.1. Bevezetés

Ez a program egy nagy nyelvi modellen alapuló chat alkalmazás, ami a 2024-es szabályrendszerben az ELTE (Eötvös Loránd Tudományegyetem) IK (Informatika Kar) PTI (Programtervező Informatikus) alapképzésére jelentkező személyeket segíti, a szabályok megismerésében, valamint a várható felvételi eredményeik kiszámolásában.

A program három fő egységből áll, ezek a felhasználói felület (3.3), a kérdés feldolgozó egység (3.4), valamint a karbantartó alkalmazások (3.5). A felhasználói felületen lehetőséget biztosítunk a nyelvi modellel való beszélgetésre, a beszélgetés szerkesztésére, valamint egyes dokumentumokban való böngészésre. A kérdés feldolgozását egy nagy nyelvi modell végzi, ami tud forrás dokumentumok között keresni (3.4.2), valamint különböző eszközöket használni (3.4.4) amennyiben szükség van rá. A karban tartó programok pedig a főprogram aktualizálására szolgálnak, lehetővé teszik a kereshető dokumentumok frissítését, valamint a nyelvi modell által adott válaszok kiértékelését.

3.2. Nyelvi modellek

A program működésének megértéséhez elengedhetetlen a nyelvi modellek alapfokú ismerete, így ebben a pontban ezt fogjuk röviden összefoglalni.

A program két féle nyelvi modellt használ, nagy nyelvi modellt (a továbbiakban modell- illetve nyelvi modellként erre hivatkozunk.), illetve beágyazási modellt. Ez előbbi úgynevezett generatív mesterséges intelligencia, ez ugyanis minden lépésben egy megadott szövegrészletből statisztikai úton visszaadja azt a token (szórészletet), ami a legnagyobb valószínűséggel következik a szöveg után, és ezt ismétli rekurzívan a generált token szöveghez fűzésével kapott új szövegre amíg egy "szöveg vége" token nem generál, vagy ki nem fogy a rendelkezésre álló tokenekből. A szöveg után következő tokenekhez a tanítási folyamat során látott több petabyte-nyi szövegben látott nyelvi minták alapján rendel valószínűségi értéket. A beágyazási modell ennél egyszerűbben működik, ez egyszerűen minden tokenhez egy sűrű vektor reprezentációt rendel, ami szemantikai információt tartalmaz, így lehetővé téve különböző szövegek jelentésbeli kapcsolatának vizsgálatát.

A programcsomag a különböző nyelvi modellek nagyrészét az OpenAI API-ján [3] keresztül éri el. A használt beágyazó modell a `text-embedding-3-large`. A nagy nyelvi modellek esetén többfélét is használunk. Rendelkezésünkre áll a GPT-3.5 Turbo valamint a GPT-4 Turbo modell is, az előbbi egyszerűbb generálási feladatok elvégzésére alkalmas azonban az eszköz használati képességei limitáltak, míg az utóbbi sokkal széleskörűbb használatra is alkalmas, viszont költségesebb a használata.

A nagy nyelvi modellek két különböző verzióját is használjuk, az egyik a `simachat` funkciókra optimalizált modell, a másik pedig az `Instruct` modell, amit arra optimalizáltak, hogy egy feladatot leíró szöveget szimplán csak folytasson.

A chat modellek bemenete JSON [8], ahol egy listában adjuk át a korábbi üzeneteket. Egy üzenetnek két adattagja van: a beosztás (`"role"`), illetve a tartalom (`"content"`), az előbbi az üzenetet küldő személyt írja le, ez lehet felhasználó (`"user"`), asszisztens (`"assistant"`), vagy rendszer (`"system"`), míg az utóbbi az üzenet szövegét tartalmazza. Itt a felhasználó és az asszisztens üzenetei a felhasználótól, valamint a modelltől jönnek, a rendszer-üzenet pedig instrukciókat tartalmaz. Ezek a modellek válaszul tudnak `simachat`, valamint JSON [8] formátumú szöveget is generálni.

Az `Instruct` modellek ennél egyszerűbben, egy `simachat` szöveget várnak, és az általuk generált szöveg egyszerűen ennek a folytatása lesz.

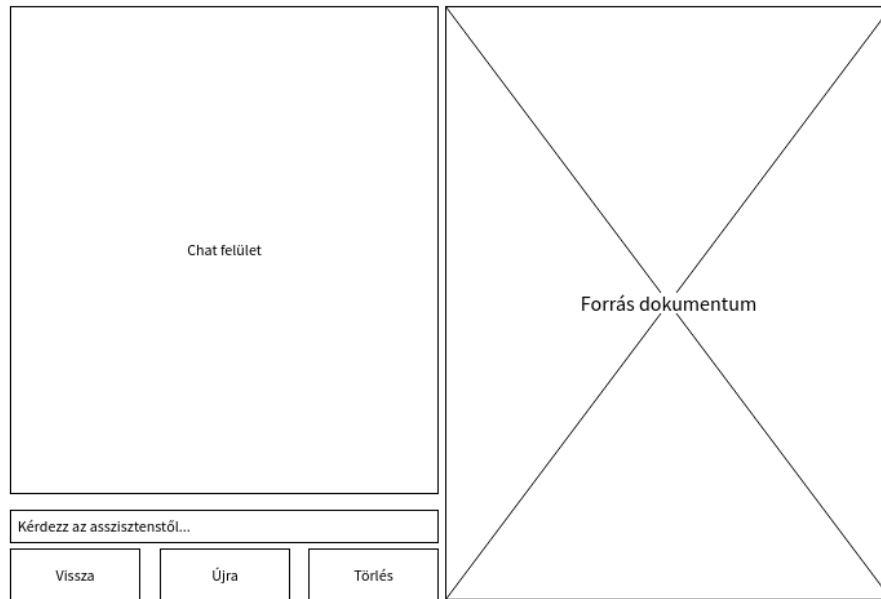
3.3. Felhasználói felület

A felhasználói felület kezeléséért a Gradio [7] felel (`gui.py`). A felhasználó kiszolgálásán túl ez az egység tartja nyilván a beszélgetés történetét is, amit egy kérdés-válasz párokból álló lista reprezentál.

A felhasználó üzenete megírása után elküldi azt az `Enter` billentyű megnyomásával, az üzenet ekkor bekerül a beszélgetéstörténetbe, ezt pedig továbbítjuk a szerveren lévő feldolgozó egység felé HTTP üzenet formájában. Miután a kérdés feldolgozásra került és válasz született, azt `markdown`ként formázzuk és adjuk hozzá a beszélgetéstörténethez, amit visszaküldünk a kliensnek. Ezek után megjelenítjük a frissített történetet a felületen.

Az üzenetek vezérlésére még biztosítunk különböző szerkesztő gombokat is.

- "Vissza": Ennek megnyomására eltávolítjuk az utolsó kérdés-válasz párt a történet listából.



3.2. ábra. A Felhasználói felület drótváz terve.

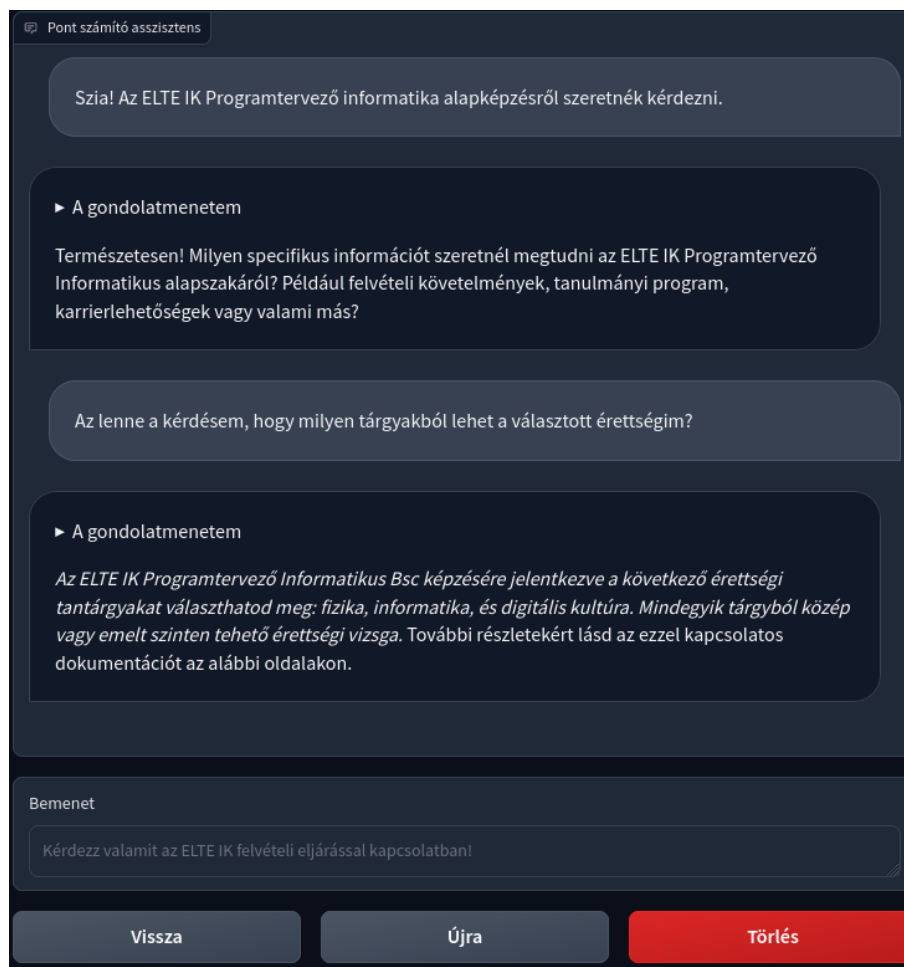
- "Újra": Ennek megnyomására visszaállítjuk a történetet az utolsó válasz előtti állapotára, azaz az utolsó elemében csak a felhasználó kérdése szerepel, és ezt újból elküldjük a feldolgozó egység felől egy új válasz generálása érdekében.
- "Törlés": Ez teljesen alapállapotra helyezi a felhasználói felület üzenetekkel foglalkozó részét. A történet egy üres lista lesz.

A válasz üzenet mellett a feldolgozó egységtől még megkapjuk a nyelvi modell gondolatmenetét is, ezt egy lenyíló menüben jelenítjük meg a markdown formázott üzenetben, a valódi válasz fölött.

```
( '<details>\n<summary>A gondolatmenetem</summary>\n<br>\n' +
  '<ul>' +
  '\n---\n'.join(['\n'.join(
    [f'<li><strong>{key}</strong>{value}</li>'
      for key, value in thought.items()])
    for thought in thinking]) +
  '</ul></details>\n\n' + response)
```

3.1. forráskód. A válasz üzenetek formázása.

A válasz pontossága érdekében a feldolgozó egység (3.4) különböző a kérdéshez kapcsolódó dokumentumokat is felhasznál a generálás során, és ezeket forrásként is jelöli. Hogy ezeket a forrásokat a felhasználó is meg tudja tekinteni hivatkozásként formázzuk és jelenítjük meg. A hivatkozás kattintás kezelését viszont fölülírjuk,



3.3. ábra. A chat felület.

hogy ne új oldalon jelenjen meg a megnyitott file hanem a felhasználói felület jobb oldalán egy `<embed>` címkében a forrásban megjelölt oldalra lapozva.

```
<a href="ik_pontszamitas.pdf#page=13">Egy generalt valasz resz
a kontextus alapjan.</a>
<a href="felvi_pontszamitas.pdf#page=2">Egy generalt valasz
resz egy masik kontextus alapjan.</a>
Egy generalt valasz resz kontextus nelkul.
```

3.2. forráskód. A források megjelölésének módja.

3.4. Feldolgozó egység

A feldolgozó egység egy chat motorból, és az ez által használt eszközökből áll. Ezek az eszközök a pontszámító eszközök, az adatbázisban való keresést végző eszköz, valamint a válasz formázó eszköz.



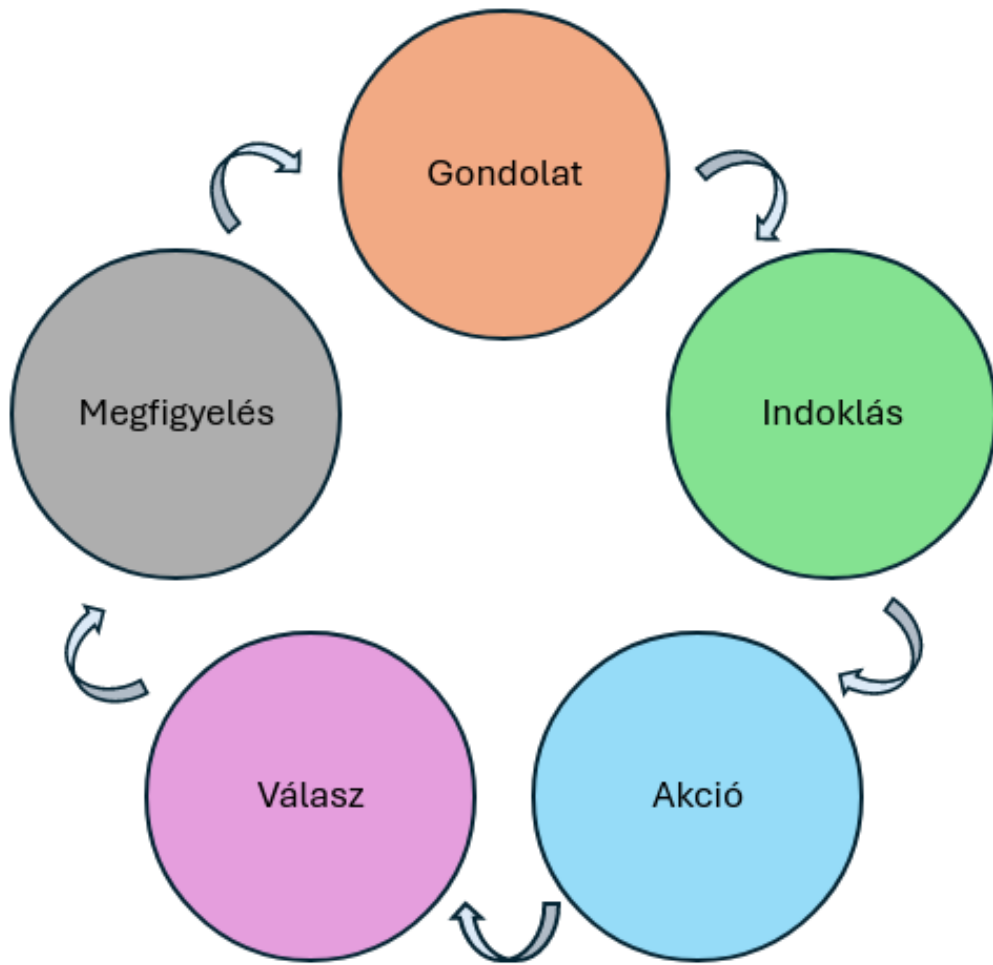
3.4. ábra. Forrás dokumentum megjelenítése.

3.4.1. Chat motor

A chat motor (`chat_engines/chat_engine.py`) egy kontroll ciklust használva működik. Ez azt jelenti, hogy amint megkap egy üzenetet ¹ a következő lépéseket hajtja végig egy ciklusban, amíg választ nem tud generálni:

- "gondolat": A nyelvi modell megformáz egy gondolatot a felhasználó kérdésével kapcsolatban. Pl.: mit tartalmaz a kérdés, mire lenne szükség a válaszhoz, meg lehet-e azonnal válaszolni, stb.
- "indoklás": A modell megindokolja, hogy miért az volt a gondolata, ami. Ebben a szakaszban fogja pontosan megfogalmazni, hogy mit is kell csinálnia az aktuális ciklusban, hogy előrébb jusson.

¹Az üzenet mellé természetesen kap üzenet történetet is, amit már a megfelelő üzenet listává konvertálunk (3.2)



3.5. ábra. A kontroll ciklus lépései egymás után.

- "akció": Annak az eszköznek a neve, amit használnia kell. Ez lehet a pontszámító, a kereső vagy akár a válasz formázó is. Amennyiben az utóbbit választotta, ez lesz az utolsó iteráció, hiszen kész a válasz.
- "válasz": Ebben a szakaszban fogja megadni a szükséges paramétereket az eszköznek, amit választott. A pontszámító esetében ezek a megfelelő eredmények, amiket bekért a felhasználótól, a kereső esetén ez üres lesz, a válasz esetén pedig a generált válasz szövege a megfelelő formátumban.
- "megfigyelés": Ezt az elemet már nem a nyelvi modell generálja, mivel itt találhatóak a használt eszköz által vissza adott adatok amennyiben az nem a válasz eszköz volt.

Minden egyes iteráció után az adott gondolatmenet hozzá lesz fűzve az eddigi beszélgetéstörténethez olyan formában, hogy az asszisztens üzenetében az iteráció

összes lépése szerepel, míg a felhasználó üzenetében csak a megfigyelés. Ezáltal a következőkben a modell rendelkezésére áll a korábbi gondolatmenete, valamint az eszközök által adott válasz is.

```
{
  "thought": "Plusz informaciora van szuksegem a kerdes
             megvalaszolasahoz.",
  "reason": "A felhasznalo a felveteli eljarasrol kerdezett, es
             a valasz informaciotartalmanak pontossaga erdekeben
             erdemes a dokumentumokban keresni. Ehhez nincs szuksegem
             tovbabi informaciora a felhasznalotol.",
  "action": "rag_tool",
  "response": {},
  "observation": {
    "query": "Milyen erettsegi targyakat lehet valasztani az
             ELTE IK PTI BSC-re a kotelezokon es az idegen nyelven
             kivul?",
    "files": [
      {"file": "ik_pontszamitas.pdf", "page": 13},
      {"file": "ik_pontszamitas.pdf", "page": 14},
      {"file": "ik_pontszamitas.pdf", "page": 12},
      {"file": "felvi_pontszamitas.pdf", "page": 2},
      {"file": "felvi_pontszamitas.pdf", "page": 1}
    ]
  }
}
```

3.3. forráskód. Példa a gondolatmenet formátumára.

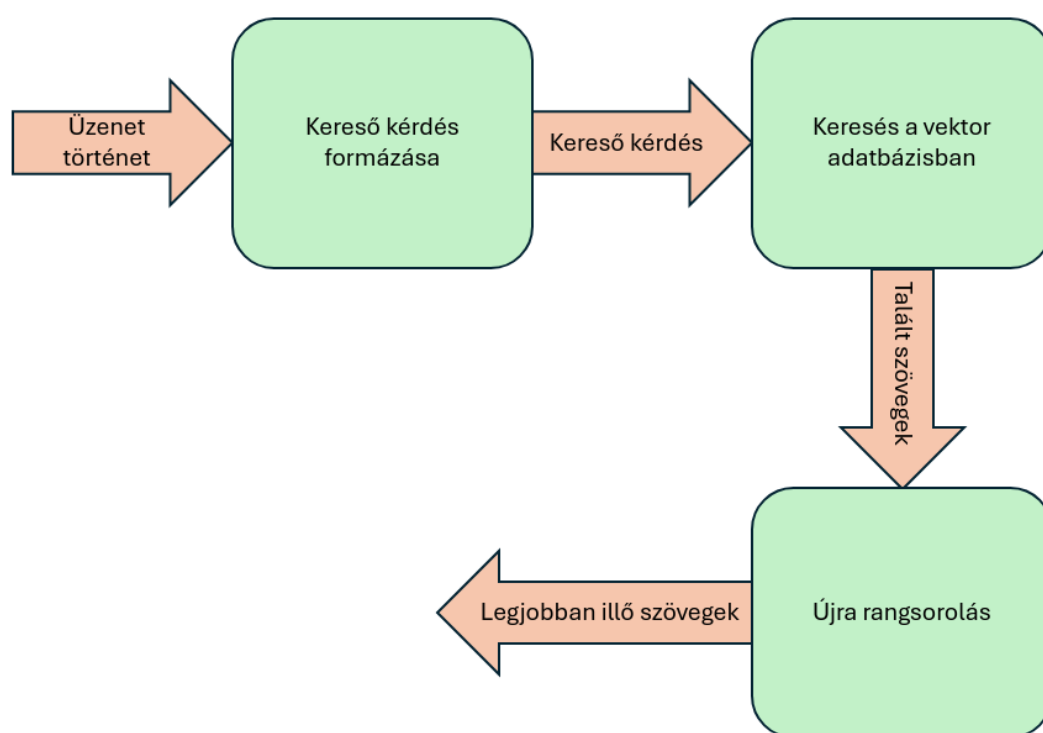
A kontroll ciklus egy meglehetősen komplex feladat ezért ehhez a GPT-4 Turbo modellt használjuk, és megadjuk neki, hogy egy JSON [8] objektum legyen a generált szöveg. Ez azzal az előnnyel is jár, hogy a modell kontextusa 128 000 token hosszú, így nem kell attól félnünk, hogy a rendszer-üzenet, vagy a beszélgetés túl hosszú lenne.

3.4.2. Kereső motor

A keresőmotort (`tools/rerank_query_engine.py`) egy eszközként tud használni a nyelvi modell, néhány apró eltéréssel.

Ugyan a kereséshez szükség van a beszélgetéstörténethez, a pontos átadás miatt ennek a helyes megadását nem bízunk a mesterséges intelligenciára, hanem mi programozzuk be. Ezért a kereső eszköznek nem kell paramétert megadni a nyelvi modell szemszögéből.

A másik eltérés, hogy a kereső eszköz által visszaadott dokumentumok tartalma külön bekerül a rendszer-üzenet egy erre kijelölt "kontextus" szekciójába, a megfigyelésbe csak a talált szövegek forrásfile-jai illetve az ezekhez tartozó oldalszámok kerülnek, valamint a kereséshez felhasznált kereső szöveg, ez jól látható a 3.3-es példában.



3.6. ábra. A keresési folyamat lépései.

A keresés menete:

- **A kérdés előfeldolgozása:** amikor a nyelvi modell, a kontroll ciklus során úgy dönt, hogy a kereső motort kell használnia bővebb kontextus szerzése érdekében, akkor az aktuális bemeneti történetet átadjuk a keresőmotornak. Ez először beadja azt egy kérdés formáló osztálynak, ami azért felel, hogy a beszélgetés történetből egy magyar, önállóan is értelmes, és általános kérdést generáljon. Ezt szintűgy egy nyelvi modell végzi, jelenleg a GPT-3.5 Turbo

Instruct [3], mivel ez nem egy chat feladat ezért lényeges hogy az Instruct modellt használjuk, ami kifejezetten az ilyen utasítások végrehajtására van optimalizálva. Erre a lépésre azért van szükség, mert az idegen nyelvű, a túl specifikus, illetve túl hosszú kereső szövegek ronthatják a keresési eredményt. Ennek a modellnek a csak 4096 token hosszú a kontextusa, így fontos, hogy limitáljuk az elküldött szöveget, ezért a beszélgetés történet utolsó 3000 tokenjét küldjük csak el.

- **Elsődleges keresés:** A keresőszöveg megszerzése után ezt felhasználva már pontosabb eredményt tudunk kapni a keresés során. Ez a folyamat egy szemantikai keresés a Weaviate [6] vektor adatbázisban, ahol a dokumentumaink előre földarabolva és beágyazva szerepelnek. Ebben a lépésben ezekből választjuk ki a keresőszöveghez szemantikailag a leg közelebb álló szöveg részleteket. A szövegrészletek mellett az adatbázistól visszkapunk különböző metaadatokat is, ezek közül a számunkra fontosak azok a forrásfile-ok és az oldalszámok ahonnan a szövegrészletek származnak.
- **Újra rangsorolás:** A releváns szövegrészletek megszerzése során ezeket nem rögtön adjuk vissza a chat motornak, hanem végzünk rajtuk még egy szűrést az INSTRUCTOR [9] beágyazó modell segítségével így pontosítva és gyorsítva a rendszer működését. Az INSTRUCTOR [9] előnye, hogy a szövegek beágyazásakor különböző instrukciókat tudunk megadni, és ez változtatni tud a tokenekhez rendelt sűrű vektoron, ami kifejezetten hasznos számunkra, hiszen nem angol nyelvű dokumentumokat szeretnénk beágyazni, és erre föl tudjuk hívni a modell figyelmét.
- Végül az újra rangsorolt szövegrészletek közül adjuk vissza a leg relevánsabbakat. A talált szövegrészleteken túl, még az első lépésben generált kérdést is vissza adjuk, főként a hibák észrevételének megkönnyítése érdekében.

```
{  
    "query": "Represent the Hungarian document for retrieval: "  
    ,  
    "corpus": "Represent the Hungarian question for retrieving  
              hungarian documents: "  
}
```

3.4. forráskód. INSTRUCTOR [9] modell utasításai a szövegek beágyazására.

Miután a keresés megtörtént és a chat motor megkapta a talált forrás szövegeket, ezeknek a tartalma, valamint a metaadataikból kinyert forrásfile-ok és oldalszámok a rendszerüzenetben fognak megjelenni a kontextus címszó alatt.

```
# Context
{
  "context": [
    {
      "content": "Forras szoveg reszlet",
      "file": "ik_pontszamitas.pdf",
      "page": 13
    }
  ]
}
```

3.5. forráskód. Példa a forrás szövegek megjelenítésére a rendszer-üzenetben.

3.4.3. Válasz formázása

A válasz generálását a nagy nyelvi modell végzi, ez az eszköz (tools/response_synthesizer.py) csak paraméterként megkapja a JSON [8] formátumban generált szöveget és ezt formázza meg a felhasználói felület elvárásainak megfelelően.

```
"action": "response_synthesizer"
"response": {
  "content": [
    {
      "text": "Egy generalt valasz resz a kontextus
              alapjan.",
      "file": "ik_pontszamitas.pdf",
      "page": 13
    },
    {
      "text": "Egy generalt valasz resz egy masik
              kontextus alapjan.",
      "file": "felvi_pontszamitas.pdf",
      "page": 2
    },
    {
      "text": "Egy generalt valasz resz kontextus nelkul.",
      "file": "None",
    }
  ]
}
```

```
        "page": "None"
    }
]
}
```

3.6. forráskód. Példa a válasz formázó paraméterezésére.

A 3.6. paraméterezésben látható formátumot az eszköz a 3.2. példában látott formátumba konvertálja.

3.4.4. Pontok kiszámolása

A nagy nyelvi modellek egy hátránya a statisztikai alapon való működés, ami azt eredményezi, hogy kifejezetten pontatlan válaszokat adnak egzakt megoldást kívánó kérdésekre (pl.: matematikai kérdések). Ezért erre a feladatra eszközöket használunk (`tools/point_calc.py`). Több különböző eszköz áll a modell rendelkezésére a különböző pont számítási módszerek létezése miatt. Az ezekkel kapcsolatos szabályozásról a <http://felvi.hu>-n lehet tájékozódni. Jelenleg két módszer van implementálva, a tanulmányi átlagot beszámító módszer: ekkor a felvételiző utolsó két év végi átlagait, illetve az összes érettségi jegyének átlagát is beleszámoljuk az összpontszámba, valamint az érettségi pontok duplázását végző módszer: ekkor csak a két releváns érettségi tárgy százalékos eredményét duplazzuk meg és így kapjuk a felvételi pontokat.

Ezen felül a felvételiző kaphat 100 pontot, ami intézményenként eltérő lehet, ez az alkalmazás csak az ELTE IK PTI Bsc szabályozását fedi le, valamint a nagy mennyiségű intézményi pont szabály miatt, csak a leggyakrabban előforduló esetek vannak implementálva, azonban kifejezett hangsúly van fektetve a kód bővíthetőségén, így igény szerint, vagy szabály változás esetén ez könnyű szerrel módosítható.

Fontos, hogy az eszközöket reprezentáló függvények jól legyenek dokumentálva a forráskódon belül, mivel a nyelvi modell ebből a dokumentációból fogja megtudni, hogy milyen eszközök állnak rendelkezésére, valamint hogy milyen információkra lesz szüksége az adott eszköz meghívásához.

```
def point_calc_double(math_final: Tuple[float, bool],
                      chosen_final: Tuple[float, bool],
                      language_exam: None | str,
                      oktv_relevant: None | int,
                      oktv_irrelevant: None | int) -> dict:
```



```
"""
    Calculates the university acceptance points for ELTE IK PTI
    BSc according to the double final exam point
    calculation
    rules.

    :param math_final: The score of the math final exam in
        percents, and a boolean which is True if the math final
        exam
    is an elevated level exam in an ordered pair.
    :param chosen_final: The score of the chosen subject final
        exam in percents, and a boolean which is True if the
        chosen subject final exam is an elevated level exam in an
        ordered pair.
    :param language_exam: Either None if the student doesn't
        have a language exam or "B2" or "C1" as strings
        indicating
        the level of language exam that they have.
    ...
```

3.7. forráskód. Példa egy eszköz dokumentációjára.

Ha a felhasználó megkéri az asszisztent, hogy számolja ki a várható felvételi pontjait, akkor annak először be kell kérni az összes szükséges információt. Elsőként az a legfontosabb, hogy a felhasználó melyik módszert szeretné használni, utána pedig, hogy ehhez a módszerhez milyen információkra lesz szüksége az asszisztensnek. Ezt a viselkedést a rendszer-üzenetben elhelyezett példák segítségével érjük el, ezt részletesebben tárgyaljuk a 3.4.5. pontban.

3.4.5. Promptok

A program nagy nyelvi modelleket használ, a kérdés keresésre való előkészítésére (3.4.2) és a chat motor (3.4.1) működéséhez. Ezeknek a modellek utasításokra van szükségük és ezeket promptok formájában, egyszerűen szöveggént tudjuk átadni nekik.

A különböző nyelvi modelleknek a promptokat szegmensekből rakjuk össze, ezek a szegmensek pedig a gyökérkönyvtár `prompts/` nevű mappájában található nyelvi modell és felhasználás szerint rendezve JSON file-okban.



3.7. ábra. Példa a felvételi pontok kiszámolására.

A promptok kiszervezésére azért történt, hogy minél kevesebb előtudással is lehessen módosítani a program működését. Például egy esetleges szabályváltozás esetén, vagy felhasználói visszajelzés alapján.

3.4.6. Speciális eljárások

Előfordulhat, hogy a chat motor (3.4.1) a kontroll ciklusában egymás után többször is ugyan azt az akciót választja, ez azért problémás mivel nem fog új információt kapni ezek válaszából, hiszen ugyan azokkal a paraméterekkel hívja meg. Legtöbbször néhány iteráció után tud választ generálni, ennek az lehet az oka, hogy a sok ismétlődés már átbillenti a döntését a válaszadásra, azonban ez nem garantált.

A végtelen ciklus elkerülésének érdekében limitálva van az iteráció szám 5-re. Ennek elegendőnek kellene lennie, hiszen a legrosszabb esetben is keres, kiszámolja

a pontokat és válaszol, de ez az eset is ritkán áll fenn, így leggyakrabban minden válasz generálása kettő vagy egy iterációból áll.

Az iterációk limitálása sajnos egy kivételt eredményez, ha elérjük a limitet, amit a felhasználó egy hiba ként fog megkapni, és újra kell generáltatnia a választ, így ezt elkerülendően, különböző instrukciókkal egészítjük ki a rendszer-üzenetet eszköz használat után. Ezek az instrukciók a `prompts/controler_engine/system_message.json` file-ban találhatók.

3.5. Karbantartó alkalmazások

A főprogram mellett két kiegészítő alkalmazást is tartalmaz a programcsomag, ezek az adatbázisba való dokumentum betöltésért, valamint a chat motor (3.4.1) válaszainak kiértékeléséért felelnek. Ezek azért kifejezetten fontosak, mert a felsőoktatási felvételi szabályai évről évre változhatnak, így egy karbantartó személynek különösebb mesterséges intelligencia ismeretek nélkül is tudnia kell frissíteni a programot.

3.5.1. Adatbázis kezelő

Az adatbázis kezelő egy CLI alkalmazás (`load_data.py`), ami lehetővé teszi dokumentumok betöltését a vektor adatbázisba, valamint ha már voltak betöltve akkor ezek felülírását. A program paraméterként kapott mappából az összes pdf dokumentumot beolvassa, majd ezt a szövegrészlet méret és a darabok közti átfedés paraméterek alapján egységes méretű tömbökre darabolja, a mondatok és paragrafusok egyben tartására figyelve. Ezek után egy beágyazó modell segítségével ezeket betölti a vektor adatbázisba. Amennyiben már korábban töltöttünk be dokumentumokat, a `-e` kapcsoló segítségével ezeket törölhetjük az újabb betöltéskor.

```
usage: load_data.py [-h] [-p PATH] [-m MODEL] [-n NAME] [-s
    CHUNK_SIZE] [-o CHUNK_OVERLAP] [-e] [-l]
```

```
Use this tool to load your pdf data quickly into the weaviate
    database for embedding and indexing.
```

```
options:
```

```
-h, --help                show this help message and exit
-p PATH, --path PATH      Path to data directory.
```

```
-m MODEL, --model MODEL
                        Open AI embedding model.
-n NAME, --name NAME   Index name.
-s CHUNK_SIZE, --chunk_size CHUNK_SIZE
                        Size of one context node in tokens.
-o CHUNK_OVERLAP, --chunk_overlap CHUNK_OVERLAP
                        Size of the overlap between context nodes
                        in tokens.
-e, --empty            Use if db already has data.
-l, --local            Use for embedded Weaviate database.
```

3.8. forráskód. Az adatbázis kezelő segítség oldala.

3.5.2. Modell kiértékelő

Az adatok változtatásával természetesen változhat a modell teljesítménye, a válaszainak pontossága. Ennek tesztelésére szolgál a kiértékelő program (`eval_model.py`). Ez az `eval/eval_qa.json`-ból beolvasott minta beszélgetés történetek alapján értékeli a modell által adott válaszokat. A beszélgetés történet felhasználó-asszisztens üzenet párokból áll, ahol az utolsó ilyen pár első eleme az utolsó üzenete a felhasználónak, a második eleme pedig az elvárt válasz. Ezen tesztesetek időnkénti frissítése szükséges, a megfelelő kiértékelés érdekében (pl.: szabályváltozás esetén).

A kiértékelés során a chat motornak (3.4.1) átadjuk az üzenet történetet a minta válasz nélkül, majd a generált válaszát a `LlamaIndex [1] CorrectnessEvaluator`-a segítségével vetjük össze a minta válasszal. Ezt a kiértékelést egy másik nyelvi modell végzi, ami egy "helyesség" számértéket rendel a válaszhoz, és amennyiben ez legalább 4.0 akkor elfogadja, ellenkező esetben elutasítja. A számérték mellé minden kiértékelését indokolja is a modell így könnyítve meg a hibák javítását.

A válasz helyességén felül még a válaszadás sebességét is vizsgáljuk.

3.6. Konténerizáció és futtatás

A futtatási környezettől való függetlenség érdekében a programcsomagot lehet Docker [10] konténerben futtatni. Ehhez elengedhetetlen, hogy minden szükséges függőség pontos verziószámmal szerepeljen a `requirements.txt` file-ban, hi-

szen a konténerben futó virtuális gépre ezek alapján lesznek telepítve a különböző Python [2] csomagok.

A vektor adatbázis egyszerűbb futtatása miatt több konténer párhuzamos futtatására van szükség, így a konténerek orkesztációját a Docker Compose [10] végzi.

Mivel a programcsomag három különálló programból áll és a karbantartó ezek bármelyikét futtathatja, ezért a konténerek indításakor egyik program sem indul el, csak a konténer interaktív módban. Ha valamelyik programot el szeretnének indítani akkor azt a konténeren belül tehetjük meg, ahogy azt a 3.9 is mutatja.

```
$ docker compose up
$ docker compose exec -it pont-IK bash
/project$ python <futtatni kívánt program>
```

3.9. forráskód. Példa a konténeren belüli futtatásra.

Lehetőség van a programok lokális futtatására, ekkor a Weaviate [6] beágyazott verziója fog futni, ezt a programon belül indítjuk el, és leáll amint a program is leáll. A lokális futtatáshoz szükséges a Python [2] 3.10-es verziója, valamint a `requirements.txt`-ben szereplő csomagok. A futtatás során a `load_data.py` valamint a `pont_ik.py` esetén a `-l` kapcsolóval tudjuk lokális módba kapcsolni őket, az `eval_model.py` esetén pedig a `USE_EMBEDDED_DATABASE` globális változó igazra állításával.

3.7. Egységtesztek

Mivel a program által használt GPT-4 Turbo és GPT-3.5 Turbo modellek nem determinisztikusak így az ezektől függő egységeket nem tudjuk tesztelni. Így csak az üzenet történet kezelése, a válasz üzenet formázása valamint a pontok kiszámolása kerül egységtesztelésre.

Szerencsére a forráskód többi részének sem kell tesztelés nélkül publikálásra kerülnie, hiszen az `eval_model.py` kiértékelőre (3.5.2) tekinthetünk egyfajta integrációs tesztként, hiszen az éles használatra nagyban hasonlító körülményeket állít elő.

3.8. Ismert hibák

A válasz generálása során felhasznált forrásokat egy `embed` HTML címke használatával jelenítjük meg, olymódon, hogy a válaszban látható hivatkozás alapvető működését felülírjuk, és kattintás esetén a `href` attribútumában szereplő URI-t megadjuk az `embed` címke `src` attribútumának.

Ez azonban Chromium alapú böngészőkben hibásan működik, mivel ilyenkor eltűnik a HTML elem és nem jelenik meg újra. A problémát a Gradio [7] által injektált JavaScript kód és a böngésző hibás interakciója eredményezheti.

3.9. Osztály szintű dokumentáció

Az osztály szintű dokumentációt automatikusan generáltuk Doxygennel, az osztály és csomag diagrammokat pedig pyreverse-el, ezek megtalálhatóak az (A) függelékben.

4. fejezet

Felhasználói dokumentáció

4.1. Karbantartó

4.1.1. Áttekintés

A programot lehet Docker Compose [10]-al, vagy konténerek használata nélkül, közvetlenül a rendszeren futtatni.

A karbantartónak két eszköz áll rendelkezésére: az adatbázis-béli dokumentumok kezelésére szolgáló CLI alkalmazás (`load_data.py`), valamint egy, a mesterséges intelligencia modell működését ellenőrző eszköz (`eval_model.py`).

Mindezen túl a főprogramot konfigurálni egy Python forrás fileban található adatosztály addattagjainak módosításával lehet.

4.1.2. Előkészületek

A programot kétféle módon lehet használni. Lehetséges közvetlenül a rendszeren, valamint Docker [10] konténerben futtatni. Az előbbi esetén a futatás előtti egyetlen dolgunk a program által használt csomagokat telepíteni.

```
$ pip install -r requirements.txt
```

4.1. forráskód. A program által használt csomagok telepítése.

A konténeres megoldás esetén nincs szükség előkészületekre.

4.1.3. Programcsomag futtatása

A program csomagot kétféle módon lehet futtatni:

- **Lokálisan**, ekkor közvetlenül a rendszeren fog futni az adatbázis, valamint a főprogram is.

```
$ python pont_ik.py -l
```

4.2. forráskód. Parancsok lokális futtatáshoz.

- **Docker Compose [10]-al**, ekkor az adatbázis, illetve a főprogram is a saját elkülönített konténerében fog futni.

```
$ docker compose up
$ docker compose exec -it pont-IK bash
/project$ python pont_ik.py
```

4.3. forráskód. Parancsok konténeres futtatáshoz.

A futtatáshoz elengedhetetlen az OpenAI api [3] kulcsának megadása. Ezt egy `.env` fájlban kell megadni a következő formátumban:

```
OPENAI_API_KEY=<api kulcs>
```

4.4. forráskód. Api kulcs megadása.

Amennyiben szükséges a szervezeti azonosítót is itt lehet megadni:

```
OPENAI_API_KEY=<api kulcs>
OPENAI_ORGANIZATION=<szervezeti azonosito>
```

4.5. forráskód. Szervezeti azonosító megadása.

Ezekhez egy példa található a `default.env` fájlban.

4.1.4. Adatbázis kezelése

Az adatbázisba lehet pdf formátumú dokumentumokat feltölteni, illetve amennyiben már korábban ez megtörtént, ezeket lehet törölni és helyettük újakat feltölteni.

Amikor feltöltünk egy dokumentumot az azonos hosszúságú részekre lesz darabolva, figyelve a mondat és paragrafus határokra, majd ezekhez a darabokhoz egy szemantikai vektor reprezentáció lesz társítva egy beágyazó modell segítségével.

A `load_data.py` futtatásánál a `-p` kapcsolóval adhatjuk meg a dokumentumok elérési útját, illetve, ha nem ez az első alkalom, hogy feltöltjük az adatbázist, akkor a `-e` kapcsolóval kiüríthetjük azt feltöltés előtt a duplikátumok elkerülése érdekében. A kapcsolók részletesebb leírásának eléréséhez a `-h` kapcsolót használhatjuk.


```
$ python load_data.py -l -e -p ./data/elte_ik
```

4.6. forráskód. Példa dokumentumok lokálisan futtatott adatbázisba töltésére.

```
$ docker compose exec pont-IK bash  
/project$ python load_data.py -e -p ./data/elte_ik
```

4.7. forráskód. Példa dokumentumok konténerben futtatott adatbázisba töltésére.

4.1.5. Mesterséges intelligencia kiértékelése

A kiértékelés egy kérdés-válasz adathalmaz alapján zajlik, ezt az eval/eval_qa.json fileban kell megadni a következő formátumban:

```
{  
  "qa_list": [  
    {  
      "history": [  
        ["Felhasznalo korabbi kerdese",  
         "Asszisztens valasza"],  
        ["Felhasznalo utolso kerdese",  
         "Elvart minta valasz"]  
      ]  
    }  
  ]  
}
```

4.8. forráskód. Kiértékelési adathalmaz megadásának módja.

A history lista utolsó elemében található a felhasználó utolsó kérdése, valamint erre az az ideális válasz, amihez az asszisztens válaszát fogjuk hasonlítani.

Az összehasonlítást egy nagy nyelvi modell végzi, ami a válasz helyességét egy helyességi értékkel méri, amennyiben ez legalább 4.0 akkor a válasz helyes. Amennyiben nem volt kielégítő az asszisztens válasza a minta válaszhoz képest, akkor ehhez a kiértékelő egy indoklást is fog társítani. A kiértékelő méri a helyesség mellett a válaszadás idejét is.

A kiértékelőt ha lokális adatbázissal szeretnénk futtatni, akkor a USE_EMBEDDED_DATABASE globális változó értékét állítsuk igazra, ellenkező esetben hamisra.

```
$ python eval_model.py
```

4.9. forráskód. Példa a modell kiértékelésére lokálisan futtatott adatbázis esetén.
label

```
$ docker compose exec pont-IK bash  
/project$ python eval_model.py
```

4.10. forráskód. Példa a modell kiértékelésére konténerben futtatott adatbázis esetén. label

4.1.6. Főprogram konfigurálása

A program konfigurálását a `configs/pont_ik_config.py` forrásfileban található `PontIkConfig` adatosztály adattagjai szerkesztésével lehet elvégezni.

PontIkConfig adattagjainak leírása

- `prompts_path`: Annak a mappának az elérési útvonala, ahol a promptokat tartalmazó JSON file-ok találhatóak.
- `db_index_name`: Az adatbázisban annak az indexnek a neve ahonnan keresni fog a nyelvi modell. Meg kell egyeznie a feltöltéskor megadottal.
- `embedding_model`: A használt OpenAI beágyazási modell neve.
- `llm_model`: A használt OpenAI nagy nyelvi modell neve.
- `question_forming_model`: A felhasználó kérdését kereséshez átalakító OpenAI modell neve.
- `question_forming_tokenizer`: A felhasználó kérdését kereséshez átalakító OpenAI modellhez tartozó tokenizer.
- `retriever_top_k`: A keresés során visszaadott dokumentum részletek száma.
- `reranker_top_n`: Az újrarangsorolás után visszaadott dokumentum részletek száma, nem lehet több a `retriever_top_k`-nél.
- `chat_history_length`: A maximális hossza az asszisztens memóriájának.

- **do_rerank**: Eldönti, hogy történjen-e újra rangsorolás. Mivel az újrarangsoroláshoz egy lokális modellt futtatunk, csak nagyobb hardware kapacitás mellett ajánlott.

Az utolsó adattag a **do_rerank** azt szabályozza, hogy a keresési folyamat során történjen-e újrarangsorolás, ez növelheti a válaszok pontosságát, azonban ehhez lokálisan kell futtatnunk egy beágyazó modellt, aminek szüksége van GPU kapacitásra.

4.1.7. Program frissítése

Mivel a felsőoktatási felvételi szabályok folyamatosan változnak így szükség van a programcsomag időnkénti aktualizálására. Ennek egy eszköze az új dokumentumok feltöltése, azonban ez esetlegesen ronthat a nyelvi modell által adott válaszok minőségén (4.1.5), vagy csak szimplán nem elegendő az új viselkedés előállítására (pl.: a pontszámítási szabályok változnak).

A karbantartónak két eszköze van arra, hogy a nyelvi modell válaszait befolyásolja, az adathalmaz aktualizálásán kívül. Ezek a promptok frissítése, valamint a pontszámító függvények módosítása.

A promptok frissítése: a gyökérkönyvtár **prompts** mappájában található felhasználás szerint darabolt, utasítás részletek a különböző nyelvi modellek működését szabályozzák. Külön mappában található a chat motor (3.4.1), a kérdés feldolgozásáért felelős nyelvi modell (3.4.2), valamint az újrarangsorolásért felelős beágyazó modell (3.4.2 utasításai. Ezek tovább vannak tagolva, almappákba valamint JSON file-okba.

```
{
  "task_description": {
    "title": "Your main task",
    "content": "You are a chat engine helping hungarian users, by
               chatting with them about the 2024 Hungarian university
               acceptance process, more specifically about the ELTE..."
  },
  "general_tool_description": {
    "title": "How to use tools",
    "content": "Your method to answer will be a simple loop, first
               you have to formulate a thought about what you want to do..."
  }
}
```

```
}...
```

4.11. forráskód. A chat motor rendszerüzenetének egy részlete. label

A nyelvi modellek számára nagyon fontosak az egyes feladatokhoz tartozó példák, így ezekből több is szerepel a rendszer üzeneteikben, ezek minden esetben külön file-t kapnak.

Fontos részlet a jelenleg használt GPT-4 illetve GPT-3.5 [3] modellekkel kapcsolatban, hogy tanítás során a rendszer üzeneteik mindig angolul szerepeltek, így ugyan a program magyarul várja el a választ tőlük, illetve a példák is magyar nyelven szerepelnek, a rendszer-üzenetekben megfogalmazott utasításoknak érdemes mindig angolul megjelenni a pontosabb működés érdekében.

A pontszámító függvények frissítése: a `tools/point_calc.py` forrásfile-ban találhatóak azok a függvények amelyek a pontok kiszámításáért felelnek, ezeket hívja meg a nyelvi modell. Jelenlegi állapotban ezek csak a gyakran előforduló eseteket tudják kezelni, így bővítés szükséges, valamint egy esetleges szabályváltozás esetén is szükséges lehet módosítani ezeket.

Amennyiben új függvényeket szeretnénk fölvenni, vagy meglévőt törölni a `pont_ik.py` forrásfile-ban a `ControllerChatEngine` osztály inicializálásánál az `other_tools` paramétert kell módosítanunk. Ez egy függvényeket tartalmazó listát vár, amiket majd a nyelvi modell tud használni.

Fontos, hogy régi függvény módosítása, illetve új függvény létrehozása esetén is a forráskódon belül a dokumentum szövegek pontosak és részletesek legyenek, ugyanis ezek alapján tudja majd ezeket meghívni a nyelvi modell. Minden paraméterről fontos leírni hogy mit csinál illetve, hogy mik lehetnek az értékei. Ha szükséges akkor a promptok közé még példákat is érdemes lehet fölvenni.

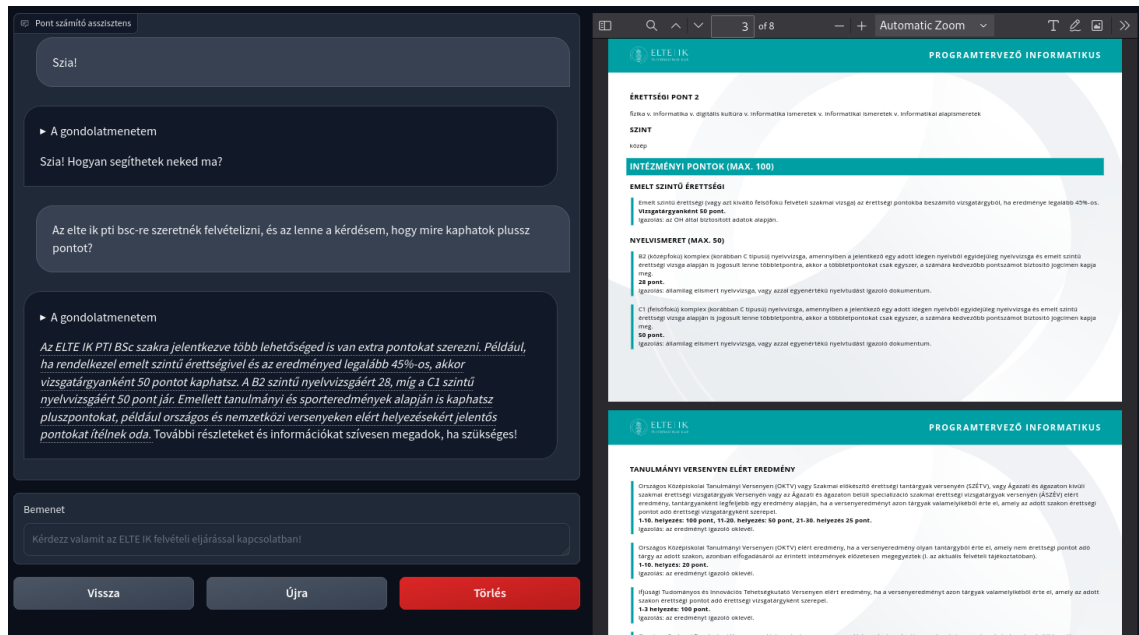
4.1.8. Futtatási követelmények

A fejlesztés során egy Intel i5-6300U CPU-n 8 GB memória mellett futtattuk a programot, és ennél gyengébb rendszeren nem ajánlott a futtatás még újrarangsorolás nélkül sem. Amennyiben szeretnénk újrarangsorolást is végezni a minimális ajánlott GPU az az Nvidia GTX 1060, ez rendelkezik elegendő videómemóriával a beágyazó modell futtatásához. A hardware-es követelmények mellett a futtatáshoz szükséges a Docker [10] vagy a Python [2] 3.10-es verziója lokális futtatás esetén,

valamint szükséges egy regisztrált felhasználó az OpenAI [3] felhasználó az API kulcsok megszerzése érdekében.

4.2. Felhasználó

4.2.1. Használat



4.1. ábra. A felhasználói felület.

A felhasználói felületen három gomb, egy bemeneti mező valamint egy üzenőfal látható. Ezen túl a felület jobb oldalán kerül megjelenítésre egy pdf-file tartalma.

A bemeneti mezőbe írhatjuk az üzeneteinket, illetve kérdéseinket az asszisztens felé, ezeket az "Enter" billentyű lenyomásával tudjuk elküldeni. Ezek az üzenetek, valamint az asszisztensről ezekre kapott válaszok kerülnek megjelenítésre az üzenőfalon. Az asszisztens válaszában tetején található egy lenyíró szövegrész, ez az asszisztens gondolatmenetét tartalmazza, ez nem releváns a válasz szempontjából, csupán betekintést nyújt a mesterséges intelligencia működésébe. A válaszüzenet szövege tartalmazhat dőlt betűs részeket, ez azt jelenti, hogy ehhez a szövegrészhez tartozik egy forrás dokumentum, a szövegrészre kattintva ezt meg tudjuk nyitni a képernyő jobb oldalán. Megnyitás után a dokumentum már a megfelelő oldalra lesz navigálva.

A felületen látható három gomb működése sorra:

- "Vissza": visszavonja az utolsó kérdésünket valamint az arra kapott választ.
- "Újra": Újra generáltatja az asszisztens legutolsó válaszát.
- "Törlés": Kiüríti az üzenőfalat, valamint a bemeneti mezőt.

Megjegyzés: Mivel a mesterséges intelligencia modell nem determinisztikus, elképzelhető, hogy a program hibára fut egy válasz generálása során. Ekkor érdemes a "Újra" gomb segítségével újra generáltatni a választ. Amennyiben újra hibára fut a program az "Vissza" gomb segítségével ajánlott átfogalmazni a kérdésünket.

4.2.2. Használati követelmények

A program egy webes eszköz, így eléréséhez internet kapcsolat szükséges. A felhasználói felület a Firefox 125.0.3-as verzióján volt a legtöbbet tesztelve, így az akadály mentes használat érdekében érdemes ezt a böngészőt használni.

Példa egy beszélgetésre az asszisztenssel

- **Felhasználó:** Tudnál segíteni kiszámolni a felvételi pontjaimat?
- **Asszisztens:** Természetesen, melyik módszert szeretnéd használni a pontok kiszámítására, az érettségi pontjaidat szeretnéd duplázni, vagy a tanulmányi átlagodat is szeretnéd beleszámítani?
- **Felhasználó:** Az érettségi pontjaimat szeretném duplázni.
- **Asszisztens:** Rendben, szükségem lenne a matek és a választott érettségi eredményedre százalékos formában, valamint hogy ezek emelt érettségik voltak-e, illetve ha oktv-n értél el helyezést azt is írd le, hogy milyen tárgyból és hányadik lettél. Ezen felül pedig azt is írd le kérlek hogy van e nyelvvizsga eredményed, és ha igen akkor milyen szintű.
- **Felhasználó:** Rendben, egy emelt matek érettségim volt 80%-os illetve egy közép infóm 60%-os, és csak egy B2-es angol nyelvvizsgám van.

- **Asszisztens:** Az általad megadott adatok alapján a várható felvételi pontszámod 318 pont. Ha úgy érzed, hogy ez nem elegendő ajánlom az emelt érettségi tételek használatát az infó érettségi során is, mivel az százalékosan is több pontot ér illetve sok intézmény plusz pontot is ad érte.

Irodalomjegyzék

- [1] Jerry Liu. *LlamaIndex*. 2022. nov. DOI: 10.5281/zenodo.1234. URL: https://github.com/jerryjliu/llama_index.
- [2] Guido Van Rossum és Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.
- [3] *OpenAI API*. URL: <https://openai.com/product>. (accessed: 03.01.2024).
- [4] Thomas Wolf és tsai. „Transformers: State-of-the-Art Natural Language Processing”. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, 2020. okt., 38–45. old. URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.
- [5] Adam Paszke és tsai. „PyTorch: An Imperative Style, High-Performance Deep Learning Library”. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, 8024–8035. old. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [6] *Weaviate Vector Database RSS*. URL: <https://weaviate.io/>. (accessed: 03.01.2024).
- [7] Abubakar Abid és tsai. „Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild”. *arXiv preprint arXiv:1906.02569* (2019).
- [8] Felipe Pezoa és tsai. „Foundations of JSON schema”. *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, 263–273. old.
- [9] Hongjin Su és tsai. „One Embedder, Any Task: Instruction-Finetuned Text Embeddings”. 2022. URL: <https://arxiv.org/abs/2212.09741>.

- [10] Dirk Merkel. „Docker: lightweight linux containers for consistent development and deployment”. *Linux journal* 2014.239 (2014), 2. old.

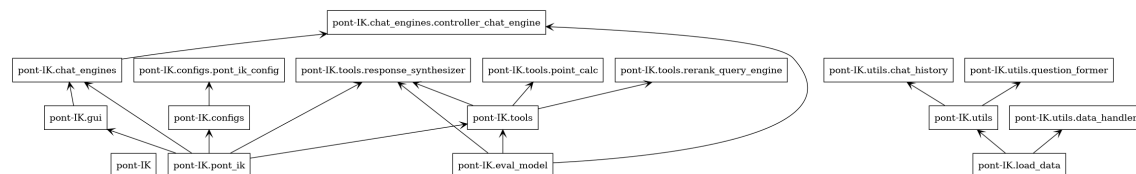
A. függelék

Osztály szintű dokumentáció

A.1. Namespace List

Here is a list of all documented namespaces with brief descriptions:

chat_engines	42
configs	42
tools	43
utils	43

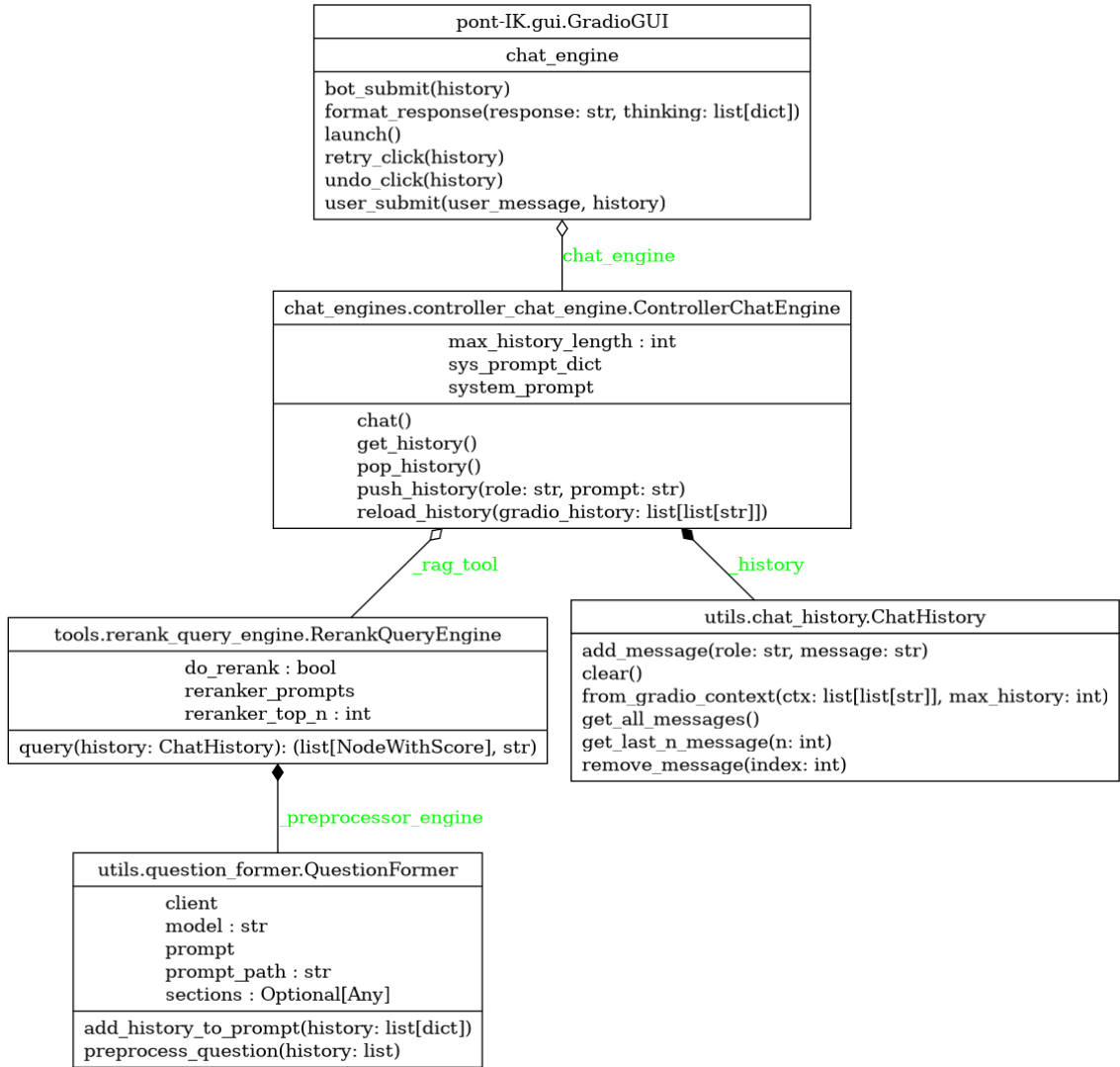


A.1. ábra. A program csomag diagrammja.

A.2. Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

chat_engines.controller_chat_engine.ControllerChatEngine	45
configs.pont_ik_config.PontIkConfig	52
eval_model.TestChatEngine	58
gui.GradioGUI	49
tools.rerank_query_engine.RerankQueryEngine	55
utils.chat_history.ChatHistory	43
utils.question_former.QuestionFormer	52



A.2. ábra. A program osztály diagrammja.

A.3. chat_engines Namespace Reference

A.3.1. Detailed Description

This module contains the classes that are used to generate responses to user queries.

A.4. configs Namespace Reference

A.4.1. Detailed Description

This module contains the configuration classes for the project.

A.5. tools Namespace Reference

A.5.1. Detailed Description

This module is a collection of tools used by the chat engine.

A.6. utils Namespace Reference

A.6.1. Detailed Description

This module contains utility functions used in the project.

A.7. utils.chat_history.ChatHistory Class Reference

Public Member Functions

- `__init__` (self)
- `add_message` (self, str role, str message)
- `remove_message` (self, int index)
- `get_all_messages` (self)
- `get_last_n_message` (self, int n)
- `clear` (self)

Static Public Member Functions

- `from_gradio_context` (list[list[str]] ctx, int max_history)

Protected Attributes

- `_history`

A.7.1. Detailed Description

A class to store chat history.

A.7.2. Constructor & Destructor Documentation

`__init__()`

```
utils.chat_history.ChatHistory.__init__ (  
    self )
```

Initialize the chat history.

A.7.3. Member Function Documentation

`add_message()`

```
utils.chat_history.ChatHistory.add_message (  
    self,  
    str role,  
    str message )
```

Add a message to the chat history.

:param role: Message sender role.

:param message: Content of the message.

`clear()`

```
utils.chat_history.ChatHistory.clear (  
    self )
```

Clear the chat history.

`from_gradio_context()`

```
utils.chat_history.ChatHistory.from_gradio_context (  
    list[list[str]] ctx,  
    int max_history ) [static]
```

Create a ChatHistory object from a Gradio context.

:param ctx: Context in the format of Gradio context.

:param max_history: The maximum number of messages to keep.

:return: The ChatHistory object.

`get_all_messages()`

```
utils.chat_history.ChatHistory.get_all_messages (
    self )
```

Get all messages in the chat history.

:return: History of messages.

`get_last_n_message()`

```
utils.chat_history.ChatHistory.get_last_n_message (
    self,
    int n )
```

Get the last `n` messages in the chat history.

:param `n`: The number of messages to get.

:return: The last `n` messages.

`remove_message()`

```
utils.chat_history.ChatHistory.remove_message (
    self,
    int index )
```

Remove a message from the chat history.

:param `index`: The index of the message to remove.

The documentation for this class was generated from the following file:

- `utils/chat_history.py`

A.8. `chat_engines.controller_chat_engine`.

ControllerChatEngine Class Reference

Public Member Functions

- `__init__` (self, `RerankQueryEngine` query_engine, Callable[[List[Dict]], str] response_tool, str prompt_path, int max_history_length, List[Callable[..., Any]] other_tools)

- `reload_history` (self, list[list[str]] gradio_history)
- `push_history` (self, str role, str prompt)
- `pop_history` (self)
- `get_history` (self)
- `chat` (self)

Public Attributes

- `max_history_length`
- `sys_prompt_dict`
- `system_prompt`

Protected Member Functions

- `str _generate_system_prompt`
(self, dict system_prompt, list[dict] examples)

Static Protected Member Functions

- `str _prompt_from_func_tool` (FunctionTool func_tool)

Protected Attributes

- `_dummy_rag_tool`
- `_rag_tool`
- `_response_tool`
- `_other_tools`
- `_chat_engine`
- `_history`

A.8.1. Detailed Description

A chat engine with a controller loop capable of using tools, searching and generating answers.

A.8.2. Constructor & Destructor Documentation

`__init__()`

```
chat_engines.controller_chat_engine.ControllerChatEngine.__init__ (
    self,
    RerankQueryEngine query_engine,
    Callable[[List[Dict]], str] response_tool,
    str prompt_path,
    int max_history_length,
    List[Callable[..., Any]] other_tools )
```

A chat engine with a controller loop capable of using tools, searching and generating answers.

:param query_engine: A pre initialized RerankQueryEngine to use for search.

:param response_tool: The tool to process the engine response.

:param prompt_path: The path to the prompt's directory.

:param max_history_length: The maximum chat history length.

:param other_tools: A list of tools for the engine to use.

A.8.3. Member Function Documentation

`_generate_system_prompt()`

```
str chat_engines.controller_chat_engine.
ControllerChatEngine._generate_system_prompt (
    self,
    dict system_prompt,
    list[dict] examples ) [protected]
```

Generates the system prompt from the system prompt dictionary and the examples.

:param system_prompt: The system prompt dictionary.

:param examples: Examples to include in the prompt.

:return: The generated system prompt.

`_prompt_from_func_tool()`

```
str chat_engines.controller_chat_engine.
ControllerChatEngine._prompt_from_func_tool (
    FunctionTool func_tool ) [static], [protected]
```


Generates a prompt from a function tool metadata.

:param func_tool: The function tool to generate the prompt from.

:return: The generated prompt.

chat()

```
chat_engines.controller_chat_engine.ControllerChatEngine.chat (
    self )
```

Generates a response according to the current chat history.

:return: The generated response, the context nodes used in the response and the thinking process of the llm as a dictionary.

get_history()

```
chat_engines.controller_chat_engine.ControllerChatEngine.get_history (
    self )
```

Returns the chat history.

:return: The chat history.

pop_history()

```
chat_engines.controller_chat_engine.ControllerChatEngine.pop_history (
    self )
```

Removes the last message from the chat history.

push_history()

```
chat_engines.controller_chat_engine.ControllerChatEngine.push_history (
    self,
    str role,
    str prompt )
```

Adds a message to the chat history.

:param role: Role of the message.

:param prompt: Content of the message.

reload_history()

```
chat_engines.controller_chat_engine.ControllerChatEngine.reload_history (
    self,
    list[list[str]] gradio_history )
```

Reloads the chat history from a gradio context.

:param gradio_history: A list of messages in the format of gradio context.

The documentation for this class was generated from the following file:

- chat_engines/controller_chat_engine.py

A.9. gui.GradioGUI Class Reference

Public Member Functions

- `__init__` (self, **ControllerChatEngine** chat_engine)
- `bot_submit` (self, history)
- `retry_click` (self, history)
- `launch` (self)

Static Public Member Functions

- `user_submit` (user_message, history)
- `undo_click` (history)
- `format_response` (str response, list[dict] thinking)

Public Attributes

- `chat_engine`
- `undo_click`
- `retry_click`
- `user_submit`
- `bot_submit`

A.9.1. Detailed Description

A class to handle the gradio GUI.

A.9.2. Constructor & Destructor Documentation

`__init__()`

```
gui.GradioGUI.__init__ (
    self,
    ControllerChatEngine chat_engine )
```

:param chat_engine: The chat engine to use when messages are sent.

A.9.3. Member Function Documentation

`bot_submit()`

```
gui.GradioGUI.bot_submit (
    self,
    history )
```

Submits the user message to the chat engine and gets the response.

:param history: The chat history in the format of gradio context.

:return: The chat history updated with the response.

`format_response()`

```
gui.GradioGUI.format_response (
    str response,
    list[dict] thinking ) [static]
```

Formats the response of the chat engine with the thinking process.

:param response: The response of the chat engine.

:param thinking: The thinking process of the chat engine.

:return: The formatted markdown response.

`launch()`

```
gui.GradioGUI.launch (
    self )
```

Defines the gradio GUI and launches the FastAPI server.

retry_click()

```
gui.GradioGUI.retry_click (
    self,
    history )
```

Removes the chat engine's last response and submits the chat history to the chat engine again.

:param history: Chat history in the format of gradio context.

:return: Updated chat history without the last assistant message and the new response from the chat engine.

undo_click()

```
gui.GradioGUI.undo_click (
    history ) [static]
```

Removes the last message from the chat history.

:param history: Chat history in the format of gradio context.

:return: Updated chat history without the last user-assistant message pair.

user_submit()

```
gui.GradioGUI.user_submit (
    user_message,
    history ) [static]
```

Adds the user message to the chat history.

:param user_message: The last message of the user from the input textbox.

:param history: The previous chat history in the format of gradio context.

:return: Empty string for the input textbox and the chat history updated with the user message.

The documentation for this class was generated from the following file:

- gui.py

A.10. configs.pont_ik_config.PontIkConfig Class Reference

Static Public Attributes

- str **prompt_path** = './prompts'
- str **db_index_name** = 'ElteIk'
- str **embedding_model** = 'text-embedding-3-large'
- str **llm_model** = 'gpt-4-turbo'
- str **question_forming_model** = 'gpt-3.5-turbo-instruct'
- str **question_forming_tokenizer** = 'cl100k_base'
- int **retriever_top_k** = 20
- int **reranker_top_n** = 5
- int **chat_history_length** = 15
- bool **do_rerank** = False

A.10.1. Detailed Description

Configuration for the ELTE IK assistant chatbot.

The documentation for this class was generated from the following file:

- configs/pont_ik_config.py

A.11. utils.question_former.QuestionFormer Class Reference

Public Member Functions

- **__init__** (self, str prompt_path, str model)
- **add_history_to_prompt** (self, list[dict] history)
- **preprocess_question** (self, list history)

Public Attributes

- `model`
- `prompt_path`
- `prompt`
- `client`
- `sections`
- `encoding`

Protected Member Functions

- `__create_prompt` (`self`, `prompt_path`)

Static Protected Member Functions

- `__message_to_string` (`dict message`)
- `__messages_to_string` (`list[dict] messages`)

A.11.1. Detailed Description

This class is responsible for forming query questions from the chat history.

A.11.2. Constructor & Destructor Documentation

`__init__()`

```
utils.question_former.QuestionFormer.__init__ (  
    self,  
    str prompt_path,  
    str model  
    str tokenizer )
```

Initialize the QuestionFormer object.

:param `prompt_path`: The path to the directory containing the prompt files.

:param `model`: The OpenAI model to use.

:param `tokenizer`: The OpenAI tokenizer that the model uses.

A.11.3. Member Function Documentation

`__create_prompt()`

```
utils.question_former.QuestionFormer.__create_prompt (
    self,
    prompt_path ) [protected]
```

Create the prompt from the prompt files.

:param prompt_path: The path to the directory containing the prompt files.

:return: The prompt.

`__message_to_string()`

```
utils.question_former.QuestionFormer.__message_to_string (
    dict message ) [static], [protected]
```

Convert a message to a string.

:param message: A message in the format of a dictionary.

:return: The message as a string.

`__messages_to_string()`

```
utils.question_former.QuestionFormer.__messages_to_string (
    list[dict] messages ) [static], [protected]
```

Convert a list of messages to a string.

:param messages: The list of messages in the format of a dictionary.

:return: The messages as a string.

`add_history_to_prompt()`

```
utils.question_former.QuestionFormer.add_history_to_prompt (
    self,
    list[dict] history )
```

Add the chat history to the prompt.

:param history: The chat history.

:return: The prompt with the chat history.

preprocess_question()

```
utils.question_former.QuestionFormer.preprocess_question (
    self,
    list history )
```

Preprocess the question.

:param history: The chat history.

:return: The generated query question.

The documentation for this class was generated from the following file:

- `utils/question_former.py`

A.12. tools.rerank_query_engine

RerankQueryEngine Class Reference

Public Member Functions

- `__init__` (self, VectorStoreIndex index, int retriever_top_k, int reranker_top_n, bool rerank, str prompt_path, str question_forming_model)
- (list[NodeWithScore], str) `query` (self, **ChatHistory** history)

Public Attributes

- `reranker_top_n`
- `do_rerank`
- `reranker_prompts`

Protected Member Functions

- str `_preprocess_query` (self, **ChatHistory** history)
- list[NodeWithScore] `_query_index` (self, str **query**)
- list[NodeWithScore] `_rerank_nodes` (self, str **query**, list[NodeWithScore] nodes)

Protected Attributes

- `_retriever`
- `_reranker`
- `_preprocessor_engine`

A.12.1. Detailed Description

A class to handle the search process of the chat engine.

A.12.2. Constructor & Destructor Documentation

`__init__()`

```
tools.rerank_query_engine.RerankQueryEngine.__init__(
    self,
    VectorStoreIndex index,
    int retriever_top_k,
    int reranker_top_n,
    bool rerank,
    str prompt_path,
    str question_forming_model
    str question_forming_tokenizer )
```

```
:param index: VectorStoreIndex to retrieve data from.
:param retriever_top_k: How many documents will be retrieved after a search.
:param reranker_top_n: How many documents will remain after reranking.
:param rerank: Whether to rerank documents.
:param prompt_path: Path to the prompt's directory.
:param question_forming_model: Open AI llm model used for
question forming before search.
:param question_forming_tokenizer: OpenAI tokenizer used by the
question forming model.
```

A.12.3. Member Function Documentation

`_preprocess_query()`

```
str tools.rerank_query_engine.RerankQueryEngine._preprocess_query (
```

```
self,  
ChatHistory history ) [protected]
```

Preprocesses the user question before searching, for a better query.

:param history: The chat history of the user and the assistant.

:return: The preprocessed user question.

`_query_index()`

```
list[NodeWithScore] tools.rerank_query_engine.RerankQueryEngine._query_index  
(  
    self,  
    str query ) [protected]
```

Retrieves the documents from the index, based on the query.

:param query: The preprocessed user question.

:return: The relevant document nodes.

`_rerank_nodes()`

```
list[NodeWithScore] tools.rerank_query_engine.RerankQueryEngine._rerank_nodes  
(  
    self,  
    str query,  
    list[NodeWithScore] nodes ) [protected]
```

Reranks the document nodes based on the query.

:param query: The preprocessed user question.

:param nodes: The retrieved document nodes.

:return: The top re ranked document nodes.

`query()`

```
(list[NodeWithScore], str) tools.rerank_query_engine.RerankQueryEngine.query  
(  
    self,  
    ChatHistory history )
```

Searches for relevant context in the database

:param history: The chat history of the user and the assistant

:return: The retrieved nodes and the preprocessed user question according to witch the search was made

The documentation for this class was generated from the following file:

- tools/rerank_query_engine.py

A.13. eval_model.TestChatEngine Class Reference

Public Member Functions

- `setup_class` (cls)
- `test_correctness` (self, subtests)

Public Attributes

- `chat_engine`
- `evaluator`

A.13.1. Detailed Description

A class to test the responses of the chat engine.

A.13.2. Member Function Documentation

`setup_class()`

```
eval_model.TestChatEngine.setup_class (  
    cls )
```

Load the necessary tools and models for the chat engine.
Create the chat engine, the query engine and the evaluator.

`test_correctness()`

```
eval_model.TestChatEngine.test_correctness (  
    self,  
    subtests )
```

Run the chat engine with the query-response pairs from the eval_qa.json file and check the correctness of the engine's responses against the example response from the file.

The documentation for this class was generated from the following file:

- eval_model.py