Szegedi Tudományegyetem

Informatikai Intézet

A nagy nyelvi modellek szemantikus képességeinek konzisztenciájának vizsgálata
Analyzing the Consistency of Semantical Capabilities of Large Language Models

Szakdolgozat

Készítette:
Fábián Bernát
informatika szakos
hallgató

Témavezetõ:
Dr. Berend Gábor
egyetemi docens

Szeged
2025

Feladatkiírás

A hallgató feladata egy olyan keretrendszer megvalósítása, amely lehetővé teszi a nagy nyelvi modellek szemantikával kapcsolatos képességeinek konzisztenciájának vizsgálatát. A kiértékelés során azt vizsgálja a keretrendszer, hogy a nagy nyelvi modellek válaszai milyen érzékenységet mutatnak olyan invarianciákra, amelyek az emberi válaszadásra nincsenek befolyással. A kísérletek során a nagy nyelvi modellek azzal kapcsolatos érzékenységének vizsgálata a cél, hogy mennyiben érzékenyek a nagy nyelvi modellek a Word-in-Context nevű feladat megoldása során az egyes inputokban szereplő mondatpárosok sorrendjének megcserélésére.

Tartalmi összefoglaló

A téma megnevezése

A nagy nyelvi modellek szemantikus képességeinek konzisztenciájának vizsgálata.

A feladat megfogalmazása

A vizsgálathoz használandó adathalmaz a Word-in-Context dataset. A kérdéseket egyenes és fordított sorrendben is fel kell tenni a modelleknek, majd összehasonlítani a válaszaikat az azonos tartalmú, de felcserélt szósorrendű kérdésekre, ezzel felmérve a szemantikus képességeiket és nyelvi konzisztenciájukat erre az emberek számára triviális, de a modellek számára problémát okozó feladat megoldására.

A megoldási mód

Akkor tekintjük helyesnek a megoldást, ha a program futtatására a kiválasztott modell egyértelmű igen/nem válaszokat ad az eldöntendő kérdésekre. A válaszainak helyessége viszont másodlagos, de nem elhanyagolható. A nyelvi modellek a Hugging Face platformról lesznek választva.

A program bemenete tetszőleges sornyi bejegyzés a Word-in-Context adathalmaz "test" splitjéből (vagy tetszőleges, ezzel megegyező formátumú kérdéshalmaz) és tetszőleges modell a Hugging Face platformról.

A megoldás kulcsa olyan szoftver fejlesztése, amely mindezt minél egyszerűbbé, automatizáltabbá és gördülékenyebbé teszi.

Az alkalmazott eszközök, módszerek

Alkalmazott eszközök:

- Google Colab és PyCharm, Python futtatókörnyezet
- Git, GitHub
- Hugging Face
- A torch, transformers, accelerate és számos egyéb Python könyvtár a Hugging Face modellek használatához

,

Alkalmazott módszerek:

- Objektumorientált programozás
- Clean Code alapelvek *Martin és Robert Clean Code* c. könyve [1] alapján

Elért eredmények

Elkészült a keretrendszer, egy Python konzolos program, amely képes tetszőleges Hugging Face modell válaszra bírását automatizálni. Egy kattintással vagy terminál paranccsal képes lefuttatni a modellt az inputként megadott adathalmazra és elvégezni a nyelvi konzisztencia tesztet, statisztikákat készítve az eredményekből.

Kulcsszavak

Nagy nyelvi modell, Word-in-Context, nyelvi konzisztencia, teljesítmény-összehasonlítás

Tartalomjegyzék

Motiváció

Az informatika és a nyelvtechnológia fejlődésével a generatív mesterséges intelligencia mindennapjaink részévé vált. A nagy nyelvi modellek kiértékelésére számos teljesítményteszt (benchmark) fejlődött ki az évek során. Az egyik legnépszerűbb ilyen teljesítményteszt a SuperGLUE Benchmark, amely 8 komoly kihívást jelentő feladat elé állítja a nagy nyelvi modelleket. A WiC (Word-in-Context) probléma a SuperGlue 8 feladatának egyike. Az emberi szöveget értő nyelvi modellek fejlesztése kiemelten fontos mind az akadémiai kutatásban, mind a gyakorlati alkalmazásokban. Az angol nyelvű szövegek feldolgozása nem csak az angolszász területeken releváns, hanem Magyarországon is, hiszen az angol nyelv használata a számítógépek világában mindennapjaink része. Az egyetem és a helyi vállalatok is aktívan foglalkoznak természetesnyelv-feldolgozási (NLP) megoldásokkal. Az AI megoldások - Hugging Face nyelvi modellek használatának - ismerete előnyt jelent mind a munkahelyeken, mind a magánéletben. IT területen különösen nagy előnyt jelent az AI megfelelő használatának ismerete, például a munkafolyamatok automatizálásában. Egy hatékony szoftver, amely képes Hugging Face modelleket futtatni, tehát nemcsak tudományos értékkel bír, hanem gyakorlati alkalmazásokban is közvetlen hasznot hozhat, főleg a nyelvészeti informatikai területen dolgozók számára.

1. fejezet

Elméleti háttér

1.1. Hasonló megoldások

Az ötlet, hogy nagy nyelvi modellek lokális telepítésére, futtatására és összehasonlítására szolgáló eszközöket hozzunk létre, nem egyedi, számos hasonló témájú és ötletű projekt született az elmúlt évtizedekben. Az alábbiakban bemutatok párat.

1.1.1. Nyelvi modellek összehasonlítását segítő eszközök

A különböző nyelvi és nem nyelvi modellek összehasonlítására egyre több projekt létezik, amelyeket nagy érdeklődés övez és nagy aktív felhasználó és fejlesztő-bázissal rendelkeznek. Például a Hugging Face Open LLM Leaderboard Model Comparator [**hf_llm_comparator**], amely elsősorban nyílt forráskódú ingyenes modellek összehasonlítására alkalmas, akár webesen, akár saját eszközre telepítve, továbbá a LMArena [5] webes felület, ahol fizetős modellek is kipróbálhatók korlátozottan, viszont csak a weboldalon keresztül. Ezeken a platformokon különböző modellek teljesítményét lehet összehasonlítani különböző feladatokon, beleértve a WiC feladatot is. Azonban ezek a platformok nem kifejezetten a nyelvi konzisztencia tesztelésére lettek kitalálva, amely a kutatásom központi eleme. A szoftver lefejlesztése előtt átelemeztem az jelenlegi legjobb módszereket és nyílt forráskódú nyelvi modelleket a Témavezetőm által javasolt LMArena és Hugging Face felületein, továbbá az utóbbiak futtatásának dokumentációját tanulmányoztam, mert azokra a szoftver elkészítéséhez szükség volt.

1.1.2. Nyelvi modellek lokális telepítését és futtatását segítő eszközök

Az utóbbi években egyre elterjedtebbek lettek azok a közösségi kezdeményezések és eszközök is, amelyek lehetővé teszik nagy nyelvi modellek helyi — internetkapcsolat nélküli vagy privát környezetben is elérhető — futtatását. Ez létfontosságú adatvédelmi, költségcsökkentési és kutatási okokból is. Ilyen eszközök:

- A Témavezetőm által ajánlott Ollama [**ollama**] CLI + desktop alkalmazás helyi API-jal sok nyílt modellt támogat.
- A szintén a Témavezetőm által ajánlott llama.cpp [**llama_cpp**] egy nyílt forráskódú C/C++ projekt.

- A szintén a Témavezetőm által ajánlott `vllm` [**kwon2023efficient**] szintén egy aktívan fejlesztett nyílt forráskódú projekt, amely elsősorban a memóriagazdálkodásra fókuszál.
- Az `LM Studio` [**lmstudio**] grafikus felület. Modellmenedzsment, többszörös modellváltás, helyi inferencia elérhető benne.
- text-generation-webui [**textgen_webui**]: webes frontend, backendként pl. `llama.cpp`, nagyon rugalmas, sokféle modellt és konfigurációt támogat.
- A `GPT4All` [**gpt4all**]: kezdeti belépő a helyi LLM-használatba — CPU-barát, alacsony küszöb, egyszerű használat jellemzi.

1.2. Az én szoftverem

A fent említett platformokhoz hasonlóan a megoldásom lehetővé teszi a nagy nyelvi modellek lokális telepítésére, futtatását, továbbá az összehasonlítását is. A projektem ugyan nem biztosít olyan kényelmes grafikus felületet, vagy a webes szolgáltatásokat, mint a legtöbb fent felsorolt projekt, ám az én projektem abban nyújt többet, hogy a Word-in-Context benchmarkon való tesztelésre sokkal alkalmasabb, mint az előbbiek, hiszen kifejezetten erre készült, továbbá aki kedveli a Python és konzolos alkalmazásokat, annak az én megoldásom kényelmesebb lehet. Ráadásul nem csak egyesével lehet beadni a modelleknek a promptokat, hanem egy futtatásra tetszőlegesen sok inferenciát kiszámíttathatunk, - olyan, minta mindegyik prompt "új chatbe" kerülne - amely jelentős előny a legtöbb felsorolt projekthez képest. A megoldásom során törekedtem arra, hogy minél több modell támogatott legyen, továbbá a keretrendszer a bárki által ingyen kipróbálható legyen, ezért ingyenesen elérhető és nyílt forráskódú eszközöket (Python, GitHub, Hugging Face) használtam.

1.3. A többértelműség problémája a természetes nyelvekben

A természetes nyelvekben a programozási nyelvekkel ellentétben egy szónak több, egymástól teljesen elkülönülő jelentése is lehet. Például az "egér" szó jelenthet egy számítógépes perifériát vagy egy állatot, és a helyes értelmezéshez a környező szavakat ismerni kell. Az ilyen jellegű többértelműségek automatikus feloldása az egyik központi problémája a természetes nyelvi rendszerek fejlesztésének. Ez az alapja a Word-in-Context feladatnak is.

1.4. A Word in context feladat

A Word in context feladatot 2019-ben fogalmazta meg Mohammad Tahmed Pilehvar, abból a célból, hogy különböző transzformer és szóbeágyazasos modelleket vizsgáljanak a feladat által megfogalmazott teljesítményteszten. A WiC feladat lényege, hogy

egy adott szó két különböző mondatbeli előfordulásáról eldöntse, hogy azonos értelemben szerepel-e. A természetes nyelvi feldolgozásban. Az ezt a feladatot megfogalmazó adathalmazt alkalmasnak találtuk a témavezetőmmel az általa megfogalmazott teljesítményteszt, a nagy nyelvi modellek szemantikus képességeinek konzisztenciájának vizsgálata elvégzéséhez. A kérdések már eleve csoportosítva vannak azonos és különböző jelentésű mondatpárokként.

A Word in context háttere

A WiC csapata alapvetően a folyamatosan fejlődő modelleknek igyekezett egy nehezebb, korszerű teljesítménytesztet állítani. Míg korábban a statikus szóbeágyazások, mint például a Word2vec és a GloVe voltak elterjedtek a szójelentés feloldására, ma már elavult módszereknek számítanak. Ezek a statikus szóbeágyazások tervezésükből adódóan nem képesek modellezni a szavak szemantikájának dinamikus természetét, vagyis azt a tulajdonságot, hogy a szavak potenciálisan különböző jelentéseknek felelhetnek meg. Egy szóhoz mindig ugyanazt a szóvektort rendelik, kontextustól függetlenül. A kontextualizált szóbeágyazások kísérletet tesznek ennek a korlátnak a feloldására azáltal, hogy dinamikus reprezentációkat számítanak ki a szavakhoz, amelyek a szövegkörnyezet alapján képesek alkalmazkodni. Ilyen szóbeágyazás transzformer például a BERT, ám ennek is megvannak a korlátai. A mai igazán modern megoldások viszont már mély tanulást és jellemzően neurális hálókat használnak a modellek szójelentés-értelmező képességeinek fejlesztésére.

1.5. Egy rendszer feladata a WiC adathalmazon

Amikor valaki kiértékelő rendszert fejleszt a WiC benchmarkra, annak feladata a szavak szándékozott jelentésének azonosítása. A WiC egy bináris osztályozási feladatként van megfogalmazva. Adott egy többjelentésű szó, amely mindkét mondatban előfordul, továbbá egy szófaj címke, kettő index és két szövegrészlet. Egy rendszer feladata, hogy meghatározza, hogy a szó ugyanabban a jelentésben használatos-e mindkét mondatban. A w célszó minden esetben csak egy ige vagy főnév lehet. A célszóhoz két eltérő szövegkörnyezet tartozik. Ezen szövegkörnyezetek mindegyike a w egy specifikus jelentését váltja ki. A feladat annak megállapítása, hogy a w előfordulásai a két szövegkörnyezetben ugyanannak a jelentésnek felelnek-e meg, vagy sem. Tehát a célszó ugyanazt a jelentést hordozza-e két különböző szövegkörnyezetben, vagy eltérőt. Ez egy összetett NLP probléma, mivel ötvözi a szójelentés-egyértelműsítés (Word Sense Disambiguation, WSD) és a kontextuális beágyazások elemeit, így a szójelentés-egyértelműsítés végrehajtásaként is értelmezhető.

1.6. Az én céljaim ehhez képest

A kutatásom első felének célja a modellek teljesítményének összehasonlítása a WiC-ből szedett kérdéseken volt, ám a többi Word-in-Context ranglétrán látható rend-

szerrel ellentétben az én célom nem az volt, hogy minél több kérdésre a gold standard [1] szerint válaszoljon egy általam tanított modell, hanem hogy megvizsgáljam, hogy mások által készített nagy nyelvi modellek válaszai milyen érzékenységet mutatnak olyan invarianciákra, amelyek az emberi válaszadásra nincsenek befolyással. Ez alapján a kérdésben a mondatok sorrendje nem szabadna, hogy befolyásolja a válaszadásukat, ám azt mégis befolyásolja. Egész pontosan a

```
Does the word `w` mean the same thing in `s1` and `s2`?
```

és a

```
Does the word `w` mean the same thing in `s2` and `s1`?
```

kérdésekre mindig ugyanazt kellene, hogy válaszolják (változatlan w, $s1$ és $s2$ esetén, ahol w egy szó, $s1$ az első példamondat, és $s2$ a második példamondat).

1.7. Bináris osztályozás

A WiC feladat egy bináris osztályozási (binary classification) problémaként van megfogalmazva: el kell dönteni, hogy egy adott szó két különböző mondatbeli előfordulása ugyanabban az értelemben szerepel-e. A bináris osztályozási feladatok problémakörében is változnak a trendek olyan szempontból, hogy egyre inkább a neurális hálók és nagy nyelvi modellek az elterjedtek bináris osztályozási feladatokra is. A Lesk-algoritmus [2] egy klasszikus szóértelem-felismerő módszer, amely a szótári definíciók és a kontextus összevetésével próbálja meghatározni a szó legmegfelelőbb jelentését. A legjobbak mégis az olyan, kifejezetten emberi szöveg megértésére specializálódott nagy nyelvi modellek, mint például a GPT-4.5 és a Gemini 3 - 2025 végén.

1.8. A WiC adathalmaz eredete

A Word in Context adathalmaz egy jó minőségű, nyelvészeti szakértők által készített benchmark adathalmaz. A mondatok a WordNetből, a VerbNetből és a Wikiszótárból származnak. A WiC adathalmaz segítségével megvizsgálhatjuk, hogy egy rendszer (modell, algoritmus) mennyire képes a szavak jelentését megérteni különböző kontextusokban. A WiC feladat része a SuperGlue [3] Benchmarknak, amely egy széles körben elfogadott benchmark nyelvi modellek kiértékelésére. 8 feladatból áll:

- BoolQ (Boolean Questions)
- CB (CommitmentBank)
- COPA(Choice of Plausible Alternatives)

[1] A gold standard egy szakértők által hitelesen és konzisztensen annotált adathalmaz, amely viszonyítási alapként szolgál automatikus rendszerek teljesítményének kiértékeléséhez.

- MultiRC (Multi-Sentence Reading Comprehension)
- ReCoRD(Reading Comprehension with Commonsense Reasoning Dataset)
- RTE(Recognizing Textual Entailment)
- **WiC(Word-in-Context)**
- WSC (Winograd Schema Challenge)

Forrás: arXiv [4]

A SuperGLUE a GLUE továbbfejlesztett változata, amelyet 2019-ben vezettek be, mivel a GLUE feladatait a legmodernebb modellek (pl. BERT) már túl jól megoldották. [4] Ez az egyik legszélesebb körben elfogadott benchmark, amelyen számos nyelvi modell teljesítményét vizsgálták. A WiC halmaz fel van osztva tanító, validációs és teszthalmazra, ezért gépi tanításra egyszerűen felhasználható. Ezt segíti elő az is, hogy az összes mondat tokenekre bontott, tabulált és egységesek az írásjelek is. A modelleket hasonlóan tanítják, mint ahogy az iskolában tanulnak a diákok. A benchmarkok feladatait jellemzően 3 részre vágják: tanító, validációs és teszthalmazra. Ennek az aránya eltérő lehet, de a tanítónak jóval nagyobbnak kell lennie, mint a másik kettőnek, és a teszthalmaznak nagyobbnak kell lennie, mint a validációs halmaznak. 80-10-10%-os eloszlás a standard, ezzel, vagy hasonló eredménnyel érhetőek általában el a legjobb eredmények.

A tanító adatbázist a korábbi iskolai példára visszatérve úgy képzelhetjük el, hogy ez a leadott anyag. A validációs halmaz olyan, mint egy mintavizsga, minta ZH. A teszthalmaz pedig a végső megmérettetés, a vizsga. Fontos, hogy a teszthalmazon sose tanítsuk a modelleket, és sose legyen átfedés a halmazok között. Ez ugyanis magoláshoz vezet, és a modell nem lesz képes általánosítani.

Szó	**Szófaj**	**Index**	**1. Példamondat**	**2. Példamondat**
defeat	N	4-4	It was a narrow defeat.	The army's only defeat.
groom	V	0-1	Groom the dogs.	Sheila groomed the horse.
penetration	N	1-1	The penetration of upper management by women.	Any penetration, however slight, is sufficient to complete the offense.
hit	V	1-3	We hit Detroit at one in the morning but kept driving through the night.	An interesting idea hit her.

1.1. táblázat. A WiC adathalmaz tesztkészletének néhány bejegyzése. *Forrás:* The Word-in-Context Dataset

[**pilehvar2019wic**]

1.8.1. Korábbi eredmények a WiC adathalmazon

A Word-in-Context honlapján található egy eredménytábla, amely bemutatja, hogy egyes modellek és algoritmusok milyen eredményt értek el a WiC feladatra. WiC adathalmazon számos modellt teszteltek, amelyek túlnyomórészt 60% feletti eredménnyel kategorizálták helyesen a mondatokat. A legjobb eredményt a SenseBERT-large rendszerrel érték el, külső erőforrások használatával. Ez az eredmény megközelíti a kézi, emberi szintű kiértékelést, melynek a felső határa 80% körüli. A kézi kiértékelésnek és a SenseBERT megoldásának egyszerűsített változatát én is elvégeztem. A kézi kiértékelésemmel közel 65, míg egy egyszerű WordNetet [**miller1994wordnet**] használó Python algoritmusommal közel 60%-os pontosságot sikerült elérnem.

Kategória	**Implementáció**	**Pontosság %**
Sentence-level contextualised embeddings		
SenseBERT-large†	Levine et al (2019)	72.1
KnowBERT-W+W†	Peters et al (2019)	70.9
RoBERTa	Liu et al (2019)	69.9
BERT-large	Wang et al (2019)	69.7
Ensemble	Gari Soler et al (2019)	66.7
ELMo-weighted	Ansell et al (2019)	61.2
Word-level contextualised embeddings		
WSDT†	Loureiro and Jorge (2019)	67.7
BERT-large	WiC's paper	65.5
Context2vec	WiC's paper	59.3
Elmo	WiC's paper	57.7
Sense representations		
LessLex	Colla et al (2020)	59.2
DeConf	WiC's paper	58.7
S2W2	WiC's paper	58.1
JBT	WiC's paper	53.6
Sentence level baselines		
Sentence Bag-of-words	WiC's paper	58.7
Sentence LSTM	WiC's paper	53.1
Random baseline (véletlenszerű kiértékelés)		
	50.0	

1.2. táblázat. Korábbi eredmények a WiC adathalmazon. *Forrás:* The Word-in-Context Dataset

[**pilehvar2019wic**]

2. fejezet

Nagy nyelvi modellek

A nagy nyelvi modell (angolul Large Language Model, LLM) olyan számítási modell, amely képes értelmes szöveg generálására vagy más természetes nyelvi feldolgozási feladatok elvégzésére. Ezek a modellek egy rendkívül költséges folyamat révén, hatalmas mennyiségű szöveges adat feldolgozásával és mélytanulási technikák alkalmazásával sajátítják el a nyelv megértését és előállítását. Az LLM-ek, mint például az OpenAI GPT-sorozata, a Google Gemini vagy a Meta LLaMA modelljei, különböző architektúrákat használnak, de leggyakrabban a transzformer alapú megközelítést alkalmazzák. Mint nyelvi modellek, az LLM-ek úgy sajátítják el ezeket a képességeket, hogy óriási mennyiségű szövegből, egy önfelügyelt és egy félig felügyelt tanulási folyamat során, statisztikai összefüggéseket tanulnak meg. Ezek a modellek képesek összetett feladatok elvégzésére, mint például szövegfordítás, összefoglalás, információ-kinyerés, kérdés-válasz és kreatív szövegalkotás. Finomhangolás által specializált feladatokra. Nagy nyelvi modelleknek jellemzően az 50 milliárd paraméteresnél nagyobb modelleket hívják. Az ennél kisebb modelleket inkább csak nyelvi modelleknek hívom.

Inputként kap egy promptot, egy legfeljebb `n` token hosszúságú szöveget, ahol az `n` függ a modelltől és a paraméterezéstől. Erre fog legfeljebb `m` hosszúságú válaszban inferenciával, azaz következtetéssel válaszolni. Azért hívják modellnek a modelleket, mert az emberi nyelvet és kommunikációt utánozzák (modellezik).

Forrás: Wikipédia

Nyelvi modell alapfogalmak

Ahhoz, hogy nagy nyelvi modelleket ki tudjam értékelni, meg kell ismerni néhány alapfogalmat, koncepciót, amely a `Hugging Face transformers` könyvtár használata során is elengedhetetlen lesz.

2.1. A prompt

A prompt alatt generatív mesterséges intelligenciák esetében egy nagy nyelvi modell számára beadott olyan inputot értünk, amelyre a modell egy kimenetet, "választ" generál, amely lehet determinisztikus (előre meghatározható) és részben véletlenszerű is. A prompt jellemzően szöveges formájú **nyelvi modellek esetén**, mint ahogy a válasz is, de

egyre elterjedtebbek a hangalapú bemeneti és kimeneti formák is. Egy jelentős előnye a promptolásnak, hogy nem csak egy előre meghatározott parancskészletet használhatunk, hanem bármit beírhatunk, nincsenek szintaktikai hibák vagy futtatási hibák, mivel a rendszer minden bemenetre képes választ generálni.
Forrás: Wikipédia

2.2. Inferencia

Egy modell promptra adott válaszát inferenciának nevezzük, ami következtetést jelent. Általánosabban az inferencia azt a folyamatot jeleni, amikor a rendszerek a tanult adatok alapján következtetéseket vonnak le képesek és új döntéseket hozni.

Ez arra utal, hogy a modell a megtanult minták alapján futásidőben logikailag következtet a legvalószínűbb kimenetre, bonyolult statisztikai számítással, nem pedig csak szó szerint "visszadobva" a betanított adatokat.
Forrás: Wikipédia

2.3. Determinisztikus következtetés, hőmérséklet, top-p és top-k

A modellek teljesítményének vizsgálata könnyebb lehet, ha a futtatás determinisztikus. Azt jelenti, hogy a modell mindig ugyanazt a kimenetet adja ugyanazon bemenet esetén, mivel a véletlenszerűséget kizáró paraméterekkel (pl. $temperature = 1.0$, $top - k = 0$, $top - p = 1.0$) működik. A hőmérséklet (temperature) a nyelvi modellek válaszainak kreativitását szabályozza (magas értéknél változatosabb, alacsonynál determinisztikusabb kimenet), a top-k pedig korlátozza a következő szó választékát a k legvalószínűbb tokenre, míg a top-p (más néven nucleus sampling) dinamikusan választja ki a valószínűségi eloszlás egy részhalmazát (pl. a legvalószínűbb tokeneket, amelyek összege eléri a p küszöböt) [9].

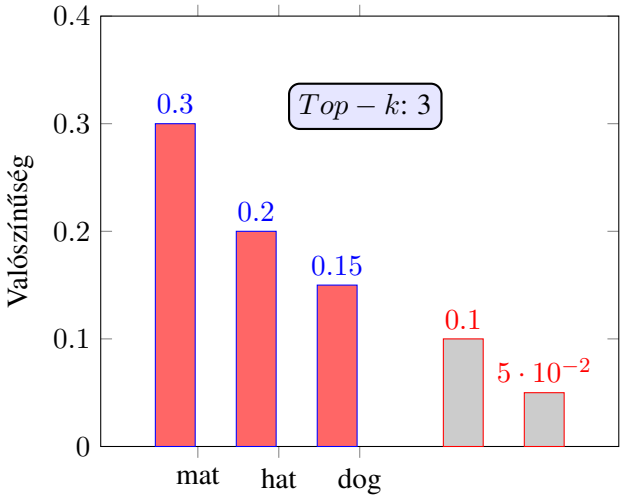
Tegyük fel, hogy a nyelvi modellnek a következő mondatot kell befejeznie:

```
The cat sat on the _ .
```

A modell célja, hogy megtalálja a legmegfelelőbb szót a hiányzó helyre. Ehhez különböző mintavételi stratégiákat alkalmazhat, például a top-k vagy top-p eljárást. A top-k módszer során a modell kiszámítja az összes lehetséges folytatás valószínűségét, majd ezek közül kiválasztja a k legvalószínűbbet, a többit pedig elveti. A kiválasztott k szóból ezt követően véletlenszerűen választ egyet, amelyet a szöveg folytatásához használ. Ez a megközelítés biztosítja, hogy csak a legrelevánsabb szavak kerüljenek figyelembe vételre, ugyanakkor némi változatosságot is megtart a generálásban.

A $top - p$ működése hasonló. Annyiban tér el, hogy nem egy fix számú legvalószínűbb szót választunk ki, hanem az odaillő szavak valószínűsége szerint. A p egy 0.00-tól 1.00-ig terjedő valós szám. A szavakat akkumulatívan vesszük számításba, amíg a p értéke nagyobb, mint a soron következő legvalószínűbb szó valószínűsége, addig belevesszük a szót és ugrunk előre. Ellenkező esetben csak belevesszük a szót, és leáll az algoritmus.

A modell szerint az alábbi 5 szó a legvalószínűbb, csökkenő eséllyel: mat, hat, dog, sofa, floor.



2.1. ábra. A top-k=3 értékadás esetén a pirossal kiemelt szavak közül fog diszkrét valószínűségi eljárás szerint választani.

Ezzel a módszerrel minél nagyobb a p érték, annál több szó közül lehet választani, de legalább 1 szót kapunk, tehát mindenképpen tudjuk folytatni a szöveget.

2.4. Maximum kimeneti tokenek és kontextusablak

A **maximum kimeneti tokenek** paraméter határozza meg, hogy a modell legfeljebb hány tokent generálhat a válasz során. Ez kizárólag a válasz hosszát korlátozza, nem befolyásolja közvetlenül a bemeneti szöveget.

A **kontextusablak** (*context window*) a modell által egyszerre figyelembe vehető tokenek teljes száma – beleértve a bemenetet és a választ is. Ha a bemenet vagy a beszélgetés túl hosszú, a legrégebbi részek elvesznek. A GPT-4 (GPT-4-0613) kontextusablaka például 8192 token. [**openai_context_window**]

3. fejezet

Google Colab

3.1. Fejlesztés Google Colabban

Az eredeti ötletem az volt, hogy a szoftvert a Google Colab környezetben valósítom meg, amely egy Google által fejlesztett webes Python-alapú futtatókörnyezet. Ehhez egy, a Témavezetőm által készített és javasolt Google Colab jegyzetfüzetet [10] fejlesztettem tovább. Ez egy egyszerű, de hatékony jegyzetfüzet, amellyel Hugging Face token birtokában tetszőleges modell válaszadásra bírható, és mindez hatékonyan automatizálható. A környezettel egy darabig hatékonyan tudtam a kiértékelő keretrendszert fejleszteni. Ám nem voltam elégedett a környezet adta korlátozásokkal, pl. session-alapú futtatókörnyezet, amely a session végével törli az adataimat, lassú csatlakozási idő a távoli hardverekhez, limitált erőforrás-hozzáférés, perzisztens adattárolási lehetőségek hiánya. Emiatt PyCharm fejlesztői környezetet használtam a keretendszer elkészítésére, lásd később 4.

4. fejezet

A szoftver: modell futtató és kiértékelő keretrendszer fejlesztése PyCharmban

A részletes kimutatásokat tartalmazó megoldás

4.1. Tervezés

A szoftvert, amelyet a szakdolgozatomhoz készítettem, 2024-ben kezdtem el fejleszteni, és azóta folyamatosan, általában heti 10-20 committal fejlesztettem. A GitHubomon elérhető a kitűzött elemek között. Korábban a https://github.com/Fabbernat/Peternity linken volt elérhető, ez most már archívum és a forráskód át lett helyezve a https://github.com/Fabbernat/Thesis linkre.

A projekt első verziója kritikákat kapott, ugyanis nehezen átlátható és nagyban AI generált, rossz minőségű kódra alapszik, így 2025 nyarán újrakezdtem a fejlesztést. Az új modulnak 2 fő fejlesztési szempontja volt.

Az egyik, hogy mind a 3 részfeladat egy kattintással elvégezhető legyen a Clean Code módszereivel, tehát nagyon egyszerűen futtatható legyen. Ez a 3 részfeladat:

- A WiC adathalmazból (vagy azzal megegyező formátumú adathalmazból modell prompt elkészítése.
- A modell futtatása az elkészített prompton, válasz generálása.
- A modell outputból adatkinyerés, statisztikák készítése.

A másik fejlesztési szempont az volt, hogy a promptok alapjául szolgáló adathalmaz és a modellek nagyon egyszerűen cserélhetőek legyenek, anélkül, hogy a program egyéb részein módosítani kellene.

A projekt kutatási szempontból érdekes része így csak az újraírt `src/Framework` modulban található. A többi archívum és mellőzhető a projekt megértéséhez. A `src/Framework` 3 almodult tartalmaz, amelyek mindegyike egy önálló, de nagyon egyszerűen futtatható programot tesz ki. Egyesítve is lehet futtatni, de több indoka is van, hogy miért inkább egyesével érdemes futtatni a programokat. Az első és utolsó modul offline futtatható és nem erőforrásigényes, a hibák esélye minimális. A középső modul a fő ok, amelyben

rengeteg a hibaforrás és a nemdeterminizmus. Online API hívásokat végez és GPU-t igényel (GPU hiányában pedig hatalmas erőforrásigényt és számítási kapacitást), ráadásul sok harmadik féltől származó függvénykönyvtárra támaszkodik. Emiatt számos hálózati és memóriakezelési hibába futhat, így érdemes külön futtatni ezt a programot, és a hibákat lekezelni. A modulok futtatási sorrendje fontos.

Ezt a szoftvert 2024-ben kezdtem el fejleszteni, de a lényegi részét 2025-ben készítettem, folyamatos fejlesztéssel, hibajavítással, kísérletezéssel.

Az eredeti beadott verzióval több gond is volt. A programot nem lehetett egy belépési ponttal futtatni. Helyette nem forduló, kaotikus, gyakran egymástól teljesen független szkriptek szerepeltek rendezetlenül, amelyeknek a használatát csak én ismertem, ráadásul egy részük nem is működött, csak úgy "volt". A program egyébként is csak a modell inputjainak előkészítését tudta, a modell inferenciát és a válasz feldolgozást már manuálisan kellett elvégezni. Csak a mostani "ModelInputPreparer"-nek megegyező rész funkcionalitásával rendelkezett standard Python környezetben. A Hugging Face modell inferenciájára (a mostani "HuggingFaceModelInferencer" modullal megegyező funkcionalitás) egy Jupyter Notebookban került sor, Google Colab környezetben. A modell outputjának feldolgozása (a mostani "ModelOutputProcessor") pedig megint egy harmadik környezetben, Google Apps Scriptben került feldolgozásra.

Az újraírásnál első lépésem volt, hogy a platformot egyesítem. Most az egész szoftver egy sima Python futtatókörnyezetben fut, egyetlen könnyen megtalálható ‘main‘fájl futtatásával, ipari standard szerint az src mappában lévő globális belépési ponttal, de egyenként is lehet futtathatni a 3 modult, mindegyikhez tartozván egy ‘main.py‘belépési pont. PyCharm környezet ajánlott a szoftver futtatásához. A legnagyobb ihletet a szoftver megálmodásához a Clean Code című könyv elolvasása hozta, amely hatására átértékeltem a programozói tudásomat és a szakdolgozatomat. Láttam, hogy objektumorientált módszerrel nagy szoftvereket is átláthatóan és biztonságosan lehet fejleszteni. Megjegyeztem azt is, hogy a jó elnevezések milyen fontosak. Ezek az elvek alapján írtam meg teljesen üres vázlatból az új szakdolgozat szoftveremet. A szoftver egy keretrendszer, hiszen a szoftver keretrendszert biztosít Hugging Face modellek lokális futtatására és azok válaszainak kiértékelésére, továbbá nincsen hozzá webes, mobilos, vagy desktop grafikus felület, nincsen hostolva stb., helyette parancssorból indítható a program és a config fájlok beállításainak módosításával paraméterezhető. Én általában csak "modulok"-ként hivatkozom a keretrendszerbe, hiszen 3, egymástól jól elkülönülő feladatú és funkciójú modulból áll. Így az egyes modulok egyénileg is futtathatóak, akkor is, ha a másik kettő nem létezik, vagy meghibásodik. Mindegyik modul egy fő futtatható állományt tartalmaz, egy-egy main.py fájt egy-egy main belépési ponttal.

Felhívom a figyelmet arra, hogy mivel a szoftver folyamatosan módosul, így a legújabb verzió valósága, működése, stb. részben eltérhet a leírttól. A leírás a 2025 év végi verzióra vonatkozik és a későbbi verziókban történő eltérések miatt felelősséget nem vállalok.

ModelInputPreparer

A ModelInputPreparer inputja egy lokális adathalmaz és az elkészíteni kívánt kérdések száma. A program az adathalmazból létrehozza a megadott számú kérdést, amelyet majd a modell inputként meg fog kapni. Ez a program alapértelmezetten ’WiC adathal-

maz' 'test' splitjének rekordjaiból hozza létre a kérdéseket, de saját, megegyező formátumú kérdésekre is működik. A kérdéseket mind egyenes, mind fordított sorrendben létrehozza, amelyekkel azután a modellek konzisztenciáját vizsgáljuk. Mivel az adathalmaz alapértelmezetten 1400 rekordból áll, és rekordonként 2 kérdésünk (egyenes, fordított) van, így alap beállításokkal 2800 kérdést kapunk, de ezen igény szerint lehet módosítani.

HuggingFaceModelInferencer

A HuggingFaceModelInferencer inputként a ModelInputPreparer outputját kapja. Feladata a modell futtatása az elkészített prompton, válasz generálása.

ModelOutputProcessor

A ModelOutputProcessor feladata a modell outputból adatkinyerés, statisztikák készítése.

4.2. Fejlesztés

4.2.1. Globális scope

GlobalMain.py: ez a szkript a fő belépési pont, amely egyszerre mind a 3 modult lefuttatja.

GlobalData mappa: a GlobalMain futtatásakor használt "adatbázis", adattároló mappa. A modulok egyesével való futtatásakor nem ezt használják, hanem az adott modul gyökerében található "data" mappát. Mindegyik modulhoz tartozik ebből egy külön.

Data mappák: az egyes modulokhoz tartozó input és output fájlok tárolását szolgálja.

.in fájlok: az input fájlok egységes kiterjesztése. Ha in a kiterjesztése, akkor valamelyik fájl azt bemetként használni fogja.

.out fájlok: az output fájlok egységes kiterjesztése

4.2.2. A modulok szerkezete

Közös elemek

Mindhárom modul gyökérkönyvtárában megtalálható az alábbi 3 Python fájl.

A `config.py` a konfigurációs fájlok, melyek célja, hogy futtatás előtt módosítani lehessen az előre beállított, nagybetűsített globális változókban tárolt beállításokat, mint például a feldolgozandó adat mennyiségét, az input fájlok adatait, vagy a prompt üzenetet.

A `main.py` a fő belépési pont, itt lehet elindítani a main függvényt, amely csak elindítja programot, átadva a futást a run.py szkriptnek.

A `run.py` pedig lefuttatja a program maradék részét, az összes modul-, függvény-, osztályhívást, tehát a program vezérlését végezve, azt általában további szkriptekre átruházva.

A HuggingFaceModelInferencer modulban található egy `modelname.py` fájl is, amely egy változóban eltárolja a kiválasztott modell nevét, ezzel biztosítva, hogy bármelyik másik modulba be lehessen importálni a kiválasztott modell nevét, anélkül, hogy egyéb függőséget behúznánk.

Mindhárom modul futásideje mérve van, a `time` Python modul `perf_counter` algoritmusával mérve. A futásidő konzolos logként tekinthető meg.

4.2.3. ModelInputPreparer

Ahhoz, hogy a modellek szemantikus képességeinek konzisztenciáját megvizsgálhassuk, először el kell érni, hogy a modell sikeresen és értelmesen választ adjon. Ahhoz pedig, hogy választ adjon, szükséges a kérdések legyártása, ugyanis a Word-in-Context adathalmaz formátuma nem alkalmas generatív nyelvi modellek promptolására. Így a statisztikai elemző modul létrehozása előtt szükség volt ennek a két funkcionalitásnak a létrehozására.

Az első modul a ModelInputPreparer (modell bemenet előkészítő), amely a WiC adathalmazból létrehozza a modellnek szánt inputot, a 2800 kérdést. Ez a `WiC` adathalmaz `test` splitjének összes rekordjából létrehoz egy kérdést, mind egyenes, mind fordított sorrendben, hogy a modellek konzisztenciáját vizsgáljuk. Mivel 1400 rekordból áll, és rekordonként 2 kérdésünk (egyenes, fordított) van, így jön ki a 2800 kérdés. A létrehozás módja a következő: A WiC adathalmaz minden sora 5, tabulátorral elválasztott értéket tartalmaz:

- szó,
- szófaj,
- a szó pozíciói a mondatokban,
- az első mondat,
- a második mondat.

Ebből nekünk csak a szó, első mondat és a második mondat szükséges, a többivel nem is fogunk foglalkozni. Először is megnyitjuk a fájlt, majd beolvassuk és eltároljuk minden sorbol a szót és a két mondatot egy listában. Ezután kétszer végigmegyünk a listán. Először elkészítjük az egyenes kérdéseket, másodjára pedig a fordítottakat. A kérdéseket úgy készítjük el, hogy az angol nyelvű kérdést tartalmazó, úgynevezett Python f-sztringekbe (formázott sztringekbe) helyezzük a szót és a 2 mondatot. Végül ezt elmentjük egy fájlba, ahonnan majd a következő modul azt elérheti.

4.2.4. HuggingFaceModelInferencer

Ebben a modulban történik a konkrét modell lefuttatása az előző modulban előállított kérdéseken.

Az előző modulban láthattuk, hogy a kérdések legyártása viszonylag egyszerű fájl- és sztringfeldolgozó algoritmusokkal lehetséges. A modell futtatása viszont nem egy triviális feladat. Még kisebb méretű Hugging Face modellek futtatása is rendkívül körülményes. Ez egy internetes API-kat és rendkívül sok időt és erőforrást (CPU, GPU, RAM)

igénylő művelet. Így nem is csoda, hogy ennek a modulnak az elkészítése vette igénybe az idő túlnyomó részét a szoftver elkészítésében.

A program elindítása előtt érdemes a `config.py` és a `modelname.py` beállításait ellenőrizni és igény szerint módosítani. A modell tetszőleges lehet, de a ajánlott a `modelname.py` fájlban található `supported\_models` listában lévő modellek közül, az elöl elhelyezkedő modellek közül választani, mert ezek kompatibilisebbek a keretrendszerrel. A config fájlban lehetőség van a prompt utasítás részén változtatni, amely aztán a kérdések elé kerül a felépítésében. Ebben lehet adni, hogy hány kérdésre válaszoljon, milyen kulcsszavakkal, terjedelemben, stb. A konzolra történő kiíratások mennyiségét is itt lehet állítani.

A program első lépésként ellenőrzi, hogy az input helyes-e. Ha páratlan kérdésből áll, akkor helytelen, hiszen nem egyezik meg az egyenes és fordított kérdések száma. Ez esetben a felhasználó figyelmeztetést kap erről, és választhat, hogy kilép a programból, vagy továbblép, és a programra bízza, hogy az megpróbálja helyreállítani - 1 sor kitörlésével.

Ezután a program azt is leellenőrzi, hogy tényleg a kérdések fele-e fordított, a most már biztosan páros hosszúságú kérdéshalmazból. Ezt úgy teszi, hogy n kérdés esetén összehasonlítja az 1. és az n/2+1. kérdésben szereplő 1. és 2. mondatot, hogy azok tényleg egymás fordítottjai-e. Hogyha nem, akkor figyelmezteti a felhasználót, hogy hibát talált a kérdéssorban. A felhasználó itt szintén választhat, hogy kilép a programból, vagy továbblép potenciálisan hibás input kérdésekkel.

Ezután kezdődik a 3. féltől származó könyvtárak használata. A szükséges könyvtárak a `torch, transformers, accelerate, huggingface_hub`, de sok modell egyéb könyvtárak telepítését is kéri. Ha valamelyik hiányzik, akkor a Python fordító, vagy a programom hibaüzenetben jelzi azt. Az első szükséges csomag a torch, amellyel előkészítjük a hardvert a nagyméretű adat számolására. A torch csomag no_grad() kontextuskezelő metódusa kikapcsolja a gradiensek számolását a szálon amin a program fut. A program maradék részét ebbe a kontextusba helyezzük, mert erre egész biztosan nem lesz szükség. Ezután a transformers csomag importálása történik. Első lépés a tokenizálás, amelyhez segítséget nyújt a transformers AutoTokenizer osztálya, és annak from_pretrained metódusa, amely a promptunkat a modell számára érthető tokenekké bontja. Ezután következik a modell-specifikus rész, hiszen a különböző modelleknek eltérő az inferencia-végrehajtó logikája. Az inferencia történhet determinisztikusan és nem-determinisztikusan, ezt is a config fájlban lehet állítani. Végül pedig a modell válaszainak a fájlba írása történik, amely mellett egyéb futás közben keletkezett információk is mentésrer kerülnek külön fájlokban.

4.2.5. ModelOutputProcessor

A `ModelOutputProcessor` bemenete a HuggingFaceModelInferencerben futtatott modell válaszai. Természetesen itt is van validálás a program bemenetére. Ha a modell jól válaszolt, akkor `n` darab kérdés esetén a válaszok száma is `n` kell, hogy legyen. Továbbá a válaszoknak az első fele az egyenes, második fele a fordított kérdésekre adott válaszokat kell, hogy tartalmazza. Emiatt ebben a modulban is elvárás, hogy az input fájl páros sorból álljon. Ha mégsem, akkor a felhasználó az előző modulokhoz hasonlóan választhat, hogy kilép a programból, vagy továbblép potenciálisan hibás input kérdésekkel.

Ha a felhasználó megerősíti, hogy tovább kíván lépni, a program megpróbálja helyreállítani az emiatt keletkező anomáliákat egy, vagy több sor törlésével.

A modul hívási verme a következő:

```
main ->
    run ->
        TernaryClassifier ->
            AnswerAwareClassificationRule
            VAGY
            SentenceClassifier
        TernaryResultsProcessor
```

Tehát a main szkript meghívja a run szkriptet, amely továbbadja az irányítást a TernaryClassifier modulnak. Ez végig megy az előző modulból inputként kapott modell válaszokon és soronként eldönti, hogy a "Yes", a "No", vagy a "?" (nem egyértelmű) kategóriába sorolható-e be leginkább. Minden sor pontosan egy kategóriába soroltatik be. Az eldöntési algoritmusnak több módszere is van, ezért választható itt az `AnswerAwareClassificationRule` és a `SentenceClassifier`, amelyet a config fájlban található `ADAPTIVE_RUN` változóval lehet beállítani. Ezután a kategóriákat a `TernaryResultsProcessor` összeveti a gold standardban szereplő értékekkel, majd az összevetés eredményei alapján készülnek el a run szkriptben a végeredmények.

Konzisztencia

A program első lépésként végigmegy a válaszok sorain. Megnézi, hogy a modell mit válaszolt soronként. Ezt 3 kategóriába sorolja: `Yes`, `No` és `?`, azaz nem egyértelmű, "talán". Ez utóbbi minden olyan válasz, amelynek nem egyértelműen "igen", vagy "nem" a jelentése. Ha ez megvan, akkor összehasonlítja az egyenes és fordított kérdéseket páronként. Ez `n` kérdés esetén `n/2` összehasonlítást jelent. Az egyezések aránya adja meg a konzisztencia mértékét, amelyet százalékban fejezek ki. Pl. 50% esetén a válaszpárok fele egyezik, 90% esetén 10-ből 9 válaszpár egyezik.

Pontosság

Önmagában az, hogy a válaszok konzisztensek, nem elég, hiszen lehet konzisztensen hibás az összes válasz is, amely nem egy kívánatos eredmény. Emiatt azt is meg kell vizsgálni, hogy a válasz megegyezik-e a gold standardbeli válasszal. Ezért egy százalékban kifejezett mutatóban eltároltam az egész adathalmazra levetített pontosság mértékét is.

Konzisztens pontosság

Ez a két érték, a konzisztencia és a pontosság alapján kiszámítható a konzisztens pontosság értéke is, amelynek a képlete a következő:

$$ConsAcc = (Cons/100 * Acc/100) * 100 \tag{4.1}$$

Tehát a konzisztencia és a pontosság 0,00 és 1,00 közé normalizált értékét szorozzuk össze egymással, majd az eredményt megszorozzuk százzal.

Kiegyensúlyozott pontosság

A nyelvi modellek hajlamosak az önismétlés hibájába esni, és az is gyakori torzulás, hogy az egyik osztályt aránytalanul preferálják a másik fölött, tehát például a WiC kérdéseinek 80%-ára nemmel válaszolnak. A kiértékelés során figyelembe vettem, hogy ha mindegyik kérdésre ugyanazt a választ adja, és ezzel aránytalanul sokat eltalál, az ne érjen annyi pontot, mint ha közel fele-fele arányban tippel vegyesen. Más szóval, ha egy modell mindenre ugyanazt válaszolja, és történetesen ezzel jól lefedi az egyoldalú gold standardot, az nem jelent valódi tudást, csak szerencsés torzítást. Ha még pontosabb képet szeretnénk kapni, megvizsgálhatjuk, hogy a modell által adott "igen" és "nem" (és a "talán") válaszok aránya hogyan oszlik el. Ha azt vesszük észre, hogy az egyik túlnyomó többségben van, akkor azt úgy is vehetjük, hogy a modell torzít és preferálja az egyik választ, így azokat kevésbé kell komolyan venni. Ebből következően a képletünkbe alacsonyabb súllyal számítjuk bele ezt a kategóriát. A kiegyensúlyozott pontosság (balanced accuracy) módszerével az abszolút `true positive` arány helyett egy átlagolt `true positive` arányt adunk meg az egyes osztályokra.

$$BalAcc = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \times 100 \tag{4.2}$$

LogFile

A ModelOutputProcessorban található naplófájl, a logFile.txt tárolja a legutóbbi futtatások eredményeit. A fájlhoz append módban minden futtatás után hozzáíródnak a legutóbbi futtatás riportjai. A megjelenített információk többek között a választott modell, a jelenlegi felhasználó neve, a futtatás dátuma, és a modell válaszok kiértékelésének eredményei: a konzisztencia, pontosság, konzisztens pontosság, kiegyensúlyozott pontosság és a tp,fp,fn,tn statisztikák.

4.3. Tesztelés

4.3.1. A programok futtatása

A PyCharm IDE kínál egyszerű és felhasználóbarát futtatási módokat - a zöld háromszög ikonokat kell keresni, azzal lehet tetszőleges .py kiterjesztésű fájlt futtatni. Ez a projektem esetén is működik. A keretrendszer main.py fájlokat kell elindítani a modulok lefuttatásához. Mivel azonban a projektem kódja sok fájlból, külsős csomagokból és összetett projektszerkezetből áll, és a zöld gomb nem biztos, hogy megfelelő útvonalakon keresi majd a modulokat, a legbiztonságosabb és legprofesszionálisabb mód a terminálos futtatás. Én, mivel Windows-on fejlesztettem, a PowerShell terminált használtam fejlesztéshez és futtatáshoz és ezt ajánlom, de a Command Prompt Shell is teljesen megfelelő.

Futtatás 3 lépésben:
Elsőként klónozzuk a projekt forráskódját a gépünkre, pl. a `~\PycharmProjects\` elérési útra. Ajánlott a projekt nevét "Thesis" néven hagyni. Következő lépésként lépjünk a projekt gyökérmappájába. Adjuk ki a

```
& .\.venv\Scripts\Activate.ps1
```

parancsot a virtuális környezet aktiválásához. Majd adjuk ki a

```
pip install torch transformers accelerate huggingface_hub
```

parancsot a szükséges csomagok telepítéséhez. Ha nagyobb modelleket is szeretnénk futtatni, akkor más csomagokra is szükség lehet. Ez esetben a

```
pip install torch transformers accelerate huggingface_hub
hf_xetm optimum
```

az ajánlott parancs.

Itt kétféle opciónk van. Egyszerű mindhárom modellt lefuttató futtatásért adjuk ki a gyökérkönyvtárban (- ahol a /.git/ és az /src/ mappa van -) a

```
py -m src.Framework.globalMain
```

parancsot. Vagy ha csak egy modult szeretnénk futtatni, akkor szintén a gyökérkönyvtárból adjuk ki a

```
py -m src.Framework.<modul neve>.main
```

parancsot. A modul neve helyére a ModelInputPreparer, HuggingFaceModelInferencer, vagy ModelOutputProcessor kerülhet, tehát a tényleges parancsok

```
py -m src.Framework.ModelInputPreparer.main

py -m src.Framework.HuggingFaceModelInferencer.main

py -m src.Framework.ModelOutputProcessor.main
```

Ezzel a fenti módszerrel garantáltan megtalálja a Python a szkriptekben megadott modulokat.

Paramétert nem vár egyik modul sem. A paraméterezés helyette a config és a ".in" kiterjesztésű fájlokban történik.

Hogyha több Python verziót is használunk, akkor annak is jelentősége lehet, hogy milyen verziójú interpreterrel futtatjuk. Tehát lehet, hogy például 3.12-es verzióval nem fut le, de 3.13-assal már igen. Ez esetben ezt is expliciten meg kell adni futtatáskor egy kapcsolóval, az alábbi módon:

```
py -3.13 -m src.Framework.globalMain
```

```
py -3.13 -m src.Framework.ModelInputPreparer.main

py -3.13 -m src.Framework.HuggingFaceModelInferencer.main

py -3.13 -m src.Framework.ModelOutputProcessor.main
```

Windowson szükség lehet a

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

parancs kiadására ahhoz, hogy az operációs rendszer engedélyezze a Python szkriptek futtatását.

4.3.2. Méret- és konzisztencia-arányban megfelelő nyelvi modellek kiválasztása és összehasonlítása

3 modellnek a keretrendszeremen elért eredményeiből ábrát készítettem.

A modellek kiválasztásakor a cél az volt, hogy kellően kisméretűek legyenek ahhoz, hogy gyorsan lefussanak a lokális környezetben, ám korlátozott paraméterszámuk ellenére a lehető legjobb "konzisztens pontosságot" tudják elérni, azaz minél több kérdésre konzisztensen és a gold standard szerint válaszoljanak.

4.3.3. Hibakezelés

A programban a hibák megtalálását és lekezelését egyszerű try-catch blokkokkal [1] kezelem. Egyes hibák az eltérő környezetekből adódnak, pl. a Python interpreter eltérő mappát tekint gökérkönyvtárnak. Ezt is try-catch-csel oldottam meg, amellyel minden importot használó szkriptbe kétféle módon is megpróbálom beimportálni a függőségeket.

4.3.4. Platformfüggetlenség

Mivel a Python virtuális környezetét használtam a programom fejlesztésére és a program nem használ abszolút útvonalakat, csak relatívakat, így a szoftverem platformfüggetlennek mondható. A szoftvert 3, eltérő architektúrájú és operációs rendszerű gépen is teszteltem. Teszteltem Asus laptopon és különböző toronygépházas gépeken, Windows 10-en, 11-en és Linux Debianon is működött a konzolos futtatás.

4.3.5. Skálázhatóság, bővíthetőség

A szoftverem könnyen tovább bővíthető, hogy több és újabb modelleket is képes legyen támogatni, és terhelhető tetszőlegesen nagy mennyiségű bemenettel, anélkül, hogy lényegesebb futásidő-növekedést tapasztalnánk, vagy lényegesebben újra kellene írni a logikáját. Az inputok beolvasása, modell választás és az outputok feldolgozása során is

[1] Pythonban a try-except a hibakezelő kulcsszavak elnevezése, de a funkciójuk ugyanaz, mint a többi nyelvben ismert try-catch blokkoknak

törekedtem arra, hogy ezek nagy mérete ne jelentsen problémát, és a validálás, hibakezelés jól működjön nagy méretű, vagy érvénytelen input esetén is. Ezt elsősorban a moduláris, és függvényekbe, osztályokba és metódusokba általánosított, absztrakt felépítésnek köszönheti a szoftver. Az egyes részek csak minimálisan függnek egymástól.

4.3.6. Megjegyzés

Fontosnak tartom megjegyezni, hogy a Python keretrendszernek van egy fontos különbsége a Google Colab megoldásomhoz képest: Itt a modell alapértelmezett módon nem 2800 futtatásban válaszol egyesével a 2800 kérdésre, hanem egy futtatásban a megadott számú kérdésre. Emiatt a Colab megoldással szemben itt ha inputként megadjuk a kérdéseket egyenes és fordított sorrendben is, akkor a modell látja a modell a korábbi kérdésre adott válaszait. Ha ezt szeretnénk elkerülni, akkor kétszer kell lefuttatni a modell inferenciát, egyszer egyenes, majd fordított kérdésekkel, vagy a megfelelő beállításokat alkalmazni a config fájlokban.

Irodalomjegyzék

[1] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ, USA: Prentice Hall, 2008. ISBN: 978-0-13-235088-4.

[2] Michael E. Lesk. „Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone”. *Proceedings of the 5th Annual International Conference on Systems Documentation (SIGDOC '86)*. New York, NY, USA: ACM, 1986, 24–26. old. URL: https://dl.acm.org/doi/10.1145/318723.318728.

[3] Paul-Edouard Sarlin és tsai. *SuperGlue: Learning Feature Matching with Graph Neural Networks*. 2020. arXiv: 1911.11763 [cs.CV]. URL: https://arxiv.org/abs/1911.11763.

[4] Alex Wang és tsai. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*. 2020. arXiv: 1905.00537 [cs.CL]. URL: https://arxiv.org/abs/1905.00537.

[5] Wei-Lin Chiang és tsai. *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*. 2024. arXiv: 2403.04132 [cs.AI]. URL: https://arxiv.org/abs/2403.04132.

[6] Springboard. *OpenAI GPT-3: Everything You Need to Know*. 2023. URL: https://www.springboard.com/blog/data-science/machine-learning-gpt-3-open-ai/.

[7] Eli Tan. „When the Terms of Service Change to Make Way for A.I. Training”. *The New York Times* (2024. jún.). URL: https://www.nytimes.com/2024/06/26/technology/terms-service-ai-training.html.

[8] Thomas Wolf és tsai. „HuggingFace's Transformers: State-of-the-art Natural Language Processing”. *arXiv preprint arXiv:1910.03771* (2019). URL: https://arxiv.org/abs/1910.03771.

[9] Ari Holtzman és tsai. „The Curious Case of Neural Text Degeneration”. *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. 2020. arXiv: 1904.09751. URL: https://arxiv.org/abs/1904.09751.

[10] Gábor Berend. *11_phi.ipynb*. https://colab.research.google.com/drive/1GQRiTDNWwNPP_PPARYd1swY1Oiai9Ey_. Google Colab jegyzetfüzet. 2025. (Elérés dátuma 2025. 05. 23.).

[11] Microsoft. *Phi-4-mini-instruct*. https://huggingface.co/microsoft/Phi-4-mini-instruct. 2024.

[12] Google. *Gemma-2-2b-it*. https://huggingface.co/google/gemma-2-2b-it. 2024.

[13] Qwen. *Qwen1.5-1.8B-Chat*. https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat. 2024.

Nyilatkozat

Alulírott Fábián Bernát, programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem Informatikai Intézet Mesterséges Intelligencia Tanszékén készítettem, a programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2025. december 2.

. . . . . .
aláírás

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani a családomnak, barátaimnak, tanítóimnak, tanáraimnak és egyetemi tanáraimnak, akik végigkísértek utamon és támogattak tanulmányaim alatt. Köszönetet szeretnék továbbá nyilvánítani szüleimnek, testvéreimnek és barátaimnak, hogy mindenben támogattak.

Végül, de nem utolsó sorban köszönetet szeretnék mondani **témavezetőmnek, Berend Gábornak**, hogy konzulensként és témavezetőként segített a szakdolgozatom megírásában.

Elektronikus mellékletek

- A keretrendszer forráskódja GitHubon, a GitHub/Fabbernat/Thesis repozitóriumban található.
- A szakdolgozat pdf forráskódja a GitHub/Fabbernat/Thesis-paper repozitóriumban található.
- A modelleket futtató Google Colab jegyzetfüzetem: ezen a linken található.
- A nyelvi modellek tesztelésének és kiértékelésének régi eredményei a Generative Language Models táblázatban tekinthető meg.