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
Programtervező informatikus 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, továbbá összemérve azt a gold standard alappal.

A megoldási mód

Akkor tekintjük helyesnek a megoldást, ha a program futtatásával a kiválasztott modell a kért mennyiségű, egyértelmű igen/nem választ képes generálni az eldöntendő kérdésekre. A válaszainak helyessége másodlagos, de nem elhanyagolható. A nyelvi modelleket a Hugging Face platformról lesznek kiválasztva.

A program bemenete tetszőleges sornyi bejegyzés a Word-in-Context adathalmaz `test` splitjéből (vagy tetszőleges, ezzel megegyező formátumú kérdéshalmazból), és tetszőleges modell a Hugging Face platformról. A program során keletkező hibákat le kell kezelni.

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

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 nyelvi modellek kiértékelésére számos teljesítményteszt (benchmark) fejlődött ki az évek során. Az egyik legnépszerűbb ilyen teljesítményteszt a SuperGLUE Benchmark, amely 8, emberi nyelvi megértést igénylő nagyon nehéz érvelés-orientált feladat elé állítja a 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 - például 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. Problémafelvetés

A nyelvi modellek gyakran érzékenyek a felhasználótól kapott prompt szósorrendjére. Architektúrájukból adódóan a modellek esetén nincs garancia, hogy bizonyos, emberek számára triviális invarianciákat konzisztensen kezeljenek. Emiatt előfordul, hogy szemantikailag azonos értelmű, de más szósorrendű input promptra inkonzisztens viselkedést mutatnak. Erre fókuszál a szakdolgozatom. Hogy majd az eredményekből következtetést lehessen levonni, két hipotézist fogalmazok meg. Egyrészt valószínűsíthető és feltételezem, hogy minél nagyobb paraméterszámú egy model, annál jobban fog teljesíteni a megfogalmazott feladatra. Továbbá méretnövekedéssel történő teljesítményjavulás mértéke is kérdéses és érdekel. Másrészt azonos méretű modellek teljesítménye között is óriási különbség lehet, még akkor is, ha azok azonos célra, pl. utasítás-végrehajtásra készültek (`instruct` modellek). Ez különösen igaz lehet a különböző cégek által gyártott modellekre. Emiatt azt is feltételezem, hogy lesznek különbségek az ilyen azonos méretű, de eltérő architektúrájú modellek között. A szakdolgozatom, pontosabban a szoftverem tesztelésének eredményei ezeket a hipotéziseket vagy megerősítik, vagy megcáfolják és ezekre a feltevésekre majd a konklúzió fejezetben reagálok.

Ehhez szükség lesz egy - lehetőleg lokális környezetben futó - programra, ahol különböző modellek telepítése és használata lehetséges.

1.2. Hasonló megoldások

Az elmúlt években számtalan, a nagy nyelvi modellek szemantikus konzisztenciájával, illetve logikai stabilitásával kapcsolatos cikk jelent meg, jól mutatva, hogy a téma nagy népszerűségnek örvend. A Google Scholar például 22 100 találatot listáz az `analyzing the semantical consistency of large language models` kifejezésre, amelyeknek a java részét az elmúlt években publikálták. [1]. A törekvés, hogy generatív modellek lokális telepítésére, futtatására és összehasonlítására szolgáló eszközöket hozzunk létre nem újkeletű, hiszen egy jó modell jelentősen megkönnyíti a munkafolyamatokat, a lokalizálása pedig biztonságosabbá és olcsóbbá teszi a használatukat. Számos hasonló témájú és ötletű projekt született az elmúlt években. Az alábbiakban bemutatok párat.

1.2.1. 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 [2], amely elsősorban nyílt forráskódú ingyenes modellek összehasonlítására alkalmas, akár webesen, akár saját eszközre telepítve, továbbá a LMArena [3] 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.

1.2.2. 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 [4] CLI + desktop alkalmazás helyi API-jal sok nyílt modellt támogat.
- A szintén a Témavezetőm által ajánlott llama.cpp [5] egy nyílt forráskódú C/C++ projekt.
- A szintén a Témavezetőm által ajánlott `vllm` [6] szintén egy aktívan fejlesztett nyílt forráskódú projekt, amely elsősorban a memóriagazdálkodásra fókuszál.
- Az `LM Studio` [7] grafikus felület. Modellmenedzsment, többszörös modellváltás, helyi inferencia elérhető benne.
- text-generation-webui [8]: webes frontend, backendként pl. `llama.cpp`, nagyon rugalmas, sokféle modellt és konfigurációt támogat.
- A `GPT4All` [9]: kezdeti belépő a helyi LLM-használatba — CPU-barát, alacsony küszöb, egyszerű használat jellemzi.

1.3. Az én szoftverem előnyei korábbiakhoz képest

A fent említett platformokhoz hasonlóan a megoldásom lehetővé teszi a nagy nyelvi modellek lokális telepítését, futtatását, továbbá az összehasonlítását is, ezáltal alkalmassá téve a szoftveremet konzisztens viselkedés hiányának a megállapítására. Az én projektem ugyan nem biztosít olyan kényelmes grafikus felületet, vagy a webes szolgáltatásokat, mint a legtöbb fent felsorolt projekt, ám abban nyújt többet, hogy a Word-in-Context

benchmarkon való tesztelésre sokkal alkalmasabb, mint az előbbiek, hiszen kifejezetten erre készült. 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, mintha 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.

1.4. 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.

1.5. A Word-in-Context feladat

A Word-in-Context adathalmaz egy jó minőségű, nyelvészeti szakértők által készített benchmark adathalmaz. A mondatok a WordNetből, a VerbNetből és a Wikiszótárból származnak. Minden sor egy azonos és különböző jelentésű mondatpárt bináris osztályozási feladatát fogalmazza meg. A WiC adathalmaz segítségével megvizsgálhatjuk, hogy egy rendszer (modell, algoritmus) mennyire képes a szavak jelentését megérteni különböző kontextusokban.

A Word-in-Context feladatot 2019-ben fogalmazták meg Mohammad Tahmed Pilehvar és munkatársai, 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. Az ezt a feladatot megfogalmazó adathalmazt alkalmasnak találtuk a Témavezetőmmel az általa megfogalmazott teljesítményteszt, a nagy nyelvi modellek szemantikus képességei konzisztenciájának vizsgálata elvégzéséhez. Ez az adathalmaz ideális, mivel eleve bináris osztályozás, és a feladat a szójelentés megértésén alapul.

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.

1.6. 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.

1.7. Az én céljaim ehhez képest

Ez a SuperGlue feladat, a Word-in-Context alapján készíthetőek olyan promptok, amelyek hasznosnak bizonyulhatnak az inkonzisztens viselkedés detektálására. A kutatásom célja a témámnak megfelelően a modellek teljesítményének összehasonlítása a WiC-ből szedett kérdéseken volt, ám a többi Word-in-Context ranglétrán látható rendszerrel ellentétben az én célom nem az volt, hogy minél több kérdésre a gold standard [1] szerint válaszoljon a vizsgált modell, hanem hogy a szavak sorrendje ne befolyásolja a válaszadásukat. Más szóval megvizsgálom, hogy mások által készített nagy nyelvi modellek válaszai milyen érzékenységet mutatnak olyan invarianciákra, amelyek az emberi válaszadásra nincsenek befolyással. 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`?
```

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

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). Ezek az egyszerű logikai invariancia - a két mondat sorrendjének felcserélése - az ember számára könnyen felismerhető és belátható, hogy a kérdés értelmét nem módosítja, ám a nyelvi modelleket, főleg az alacsony, "csak" pár milliárd paraméterszámúakat könnyen összezavarja.

A fenti felcserélés egy klasszikus logikai invariancia teszt. Ha $A \wedge B$ igaz, akkor $B \wedge A$-nak is igaznak kell lennie. Ebből következően ha a modell szerint a w szó jelentése azonos $s1$ és $s2$ kontextusban, akkor az $s2$ és $s1$ sorrendben is azonosnak kell lennie.

A kérdéseket a fent látható formátumban egyenes és fordított sorrendben is fel fogom tenni a modelleknek. A továbbiakban erre a formátumra fogok "egyenes" és "fordított" kérdésekként hivatkozni. A keretrendszerem **minden esetben** így várja, és az egyenes és a fordított sorrendű kérdések száma is **minden esetben** szigorúan megegyezik. Ha mégsem teljesül a fentiek valamelyike az inputra (bemenetre), akkor a felhasználó arról feltétlenül értesítést kap.

1.8. 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 [10] 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.

1.9. A WiC adathalmaz eredete

A WiC feladat része a SuperGlue [11] Benchmarknak, amely egy széles körben elfogadott benchmark nyelvi modellek kiértékelésére.

8 feladatból áll:

- BoolQ (Boolean Questions)
- CB (CommitmentBank)
- COPA(Choice of Plausible Alternatives)
- 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 [12]

A SuperGLUE a GLUE továbbfejlesztett változata, amelyet 2019-ben azért vezettek be, mert az előző versenykörnyezet, a GLUE feladatai könnyűvé váltak a modern, nagyteljesítményű NLP-modellek számára. [12] Az új benchmark célja az volt, hogy keményebb, nehezebb nyelvi és értelmi kihívásokat állítson. 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

[13]

1.9.1. 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 [14] 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

[13]

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. Alkalmazhatóak finomhangolás által specializált feladatokra. Nagy nyelvi modelleknek jellemzően az 50 milliárd paraméteresnél nagyobb modelleket hívják. Az ennél kisebb modelleket emiatt a továbbiakban következetesen 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 a 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 olyan inputot értünk, amelyre a modell egy kimenetet, "választ" generál, amely lehet determinisztikus (előre meghatározható) és részben véletlenszerű is. A prompt jellemzően szöveges for-

májú **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 jelenti, amikor a rendszerek a tanult adatok alapján következtetéseket vonnak le és képesek új döntéseket hozni. A projektemben a generatív modellek inferenciájának folyamatára az egyszerűség kedvéért általában "a modell futattásaként" fogok hivatkozni.

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

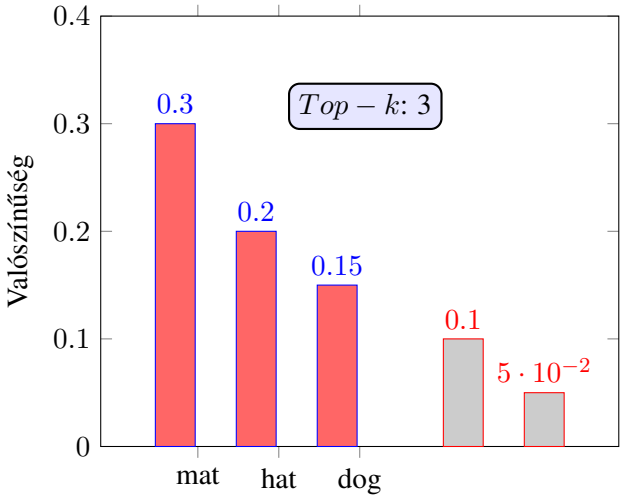
A modellek teljesítményének vizsgálata könnyebb lehet, ha a futtatás determinisztikus. Azt jelenti, hogy a modell mindig ugyanazt a kimenetet adja ugyanazon bemenet esetén, mivel a véletlenszerűséget kizáró paraméterekkel (pl. $temperature = 0.0$, $top - k = 0$, $top - p = 1.0$) működik. A hőmérséklet (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) [15].

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 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.

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. 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. [16]

2.5. Logikai szövegkövetkeztetési konzisztencia

A konzisztencia, konzisztens kifejezések sokféleképpen értelmezhetők a nagy nyelvi modellek szaknyelvezetében. Éppen ezért tisztázom, hogy szavakat a továbbiakban kifejezetten a feladatkiírásban és az 1. fejezetben meghatározott logikai és szövegkövetkeztetési értelemben használom, és nem a konzisztencia más típusaira, pl. a faktuális, önfigyelő (self-consistency), időbeli, szövegösszefoglalási, vagy tanácsadási konzisztenciára értve –, mivel ezeket eltérő módszerekkel kellene vizsgálni.

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. Ötletem megvalósításához a Témavezetőm által készített és javasolt Google Colab jegyzetfüzetet [17] fejlesztettem tovább. Ez egy egyszerű, de hatékony jegyzetfüzet, amellyel egy tetszőleges modell válaszadásra bírható. A környezettel egy darabig, a modell válaszok legeneráltatása erejéig effektíven és eredményesen tudtam a kiértékelő keretrendszert fejleszteni. Az elkészült jegyzetfüzet 6 jelenlegi formájában képes a meghatározott számú, akár mind a 2800 Word-in-Context-alapú kérdésre választ generáltatni a meghatározott modellen, így rendelkezik a keretrendszer középső moduljának 4.1 funkcionalitásával. Használata egyszerű: csak le kell futtatni sorban az összes cellát, mely akár egyetlen kattintással is elvégezhető A Google Colab egyszerű és kényelmes környezetet biztosít modellek futtatására, akár limitált ingyenes GPU-használattal is. 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 és egyéb korlátozások, amelyek csak előfizetéssel kerülhetőek ki. Emiatt PyCharm fejlesztői környezetet használtam a keretendszer elkészítésére.

4. fejezet

A szoftver: modellt futtató és kiértékelő keretrendszer fejlesztése PyCharmban

A részletes kimutatásokat tartalmazó megoldás

4.1. Tervezés

A szoftver elődjét, amelyet a szakdolgozatomhoz készítettem, 2024-ben kezdtem el fejleszteni, és azóta folyamatosan, általában heti git 10-20 committal fejlesztettem. A GitHubomon elérhető a kitűzött elemek között. A teljes forráskód az összes korábbi verzióval együtt a
https://github.com/Fabbernat/Thesis GitHub repozitóriumban található.

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

Az egyik, hogy mind a 3 részfeladat egy kattintással elvégezhető legyen a Clean Code [18] 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 kimenetből 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 nagyrészt 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 elindítani 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 `ModelInputPreparernek` megegyező rész funkcionalitásával rendelkezett standard Python környezetben. A Hugging Face modell inferenciájára (a mostani `HuggingFaceModelInferencer` 4.1 modullal megegyező funkcionalitás) egy Jupyter Notebookban került sor, Google Colab környezetben. A modell outputjának (kimenetének) feldolgozása (a mostani `ModelOutputProcessor`) pedig megint egy harmadik környezetben, Google Apps Scriptben került feldolgozásra.

Az újraírásnál első lépésem volt, hogy a platformot egyesítettem. Most az egész szoftver egy sima, standard 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` fájlban található belépési pont. PyCharm környezet ajánlott a szoftver futtatásához. A legnagyobb ihletet a szoftver megálmodásához a Clean Code [18] című könyv elolvasása hozta, amely hatására átértékeltem a programozói tudásomat és a szakdolgozatomat. Láttam, hogy modulárisan, objektumorientált módszerrel és az egymásra halmozódó függőségek minimalizálásával 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 keretrendszert biztosít Hugging Face modellek lokális futtatására és azok válaszainak kiértékelésére, továbbá nincsen hozzá webes, mobilos, vagy desktop grafikus felület és nincsen hostolva sem. Helyette parancssorból indítható a program és a `config` fájlok beállításainak módosításával paraméterezhető. Én általában csak "modulokként" hivatkozom a keretrendszerre, hiszen három, egymástól jól elkülönülő feladatú és funkciójú modulból áll. Így az egyes modulok egyedülállóan 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 szoftvert időnként módosul, így a legújabb GitHub verzió konfigurációi, paraméterei, működése stb. részben eltérhetnek a leírttól. A leírás a 2025 év végi verzióra vonatkozik és a későbbi verziókban kisebb eltérések előfordulhatnak. A legtöbb eltérés korábbi verziók visszaállításával, vagy a config és az input fájlok módosításával kiküszöbölhető.

ModelInputPreparer

A `ModelInputPreparer` modul inputjai egy lokális adathalmaz és az elkészíteni kívánt kérdések száma. A program feladata a WiC adatokból a megadott számú az

egyenes és fordított kérdés legyártása, amelyet majd a nyelvi modell inputként meg fog kapni. Ez a program alapértelmezetten `Word-in-Context` adathalmaz `test` splitjének rekordjaiból hozza létre a kérdéseket, de saját, megegyező formátumú kérdésekre is működik. A ModelInputPreparer alapértelmezetten randomizálja az elkészített kérdések sorrendjét a tiszta kiértékelés érdekében. 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. Itt hajtódik végre a távoli Hugging Face modell(ek) letöltése, telepítése és futtatása. Feladata a modell futtatása az elkészített prompton, válasz generálása. A generálási stratétiám lényege, hogy minden inferencia során 1 kérdést válaszoltatok meg, egymástól függetlenül, így elkerülve a modellek "emlékezőképességét", továbbá csak 1 output tokent engedélyezek, biztosítva, hogy a válasz csak egy "igen" vagy "nem" lesz.

ModelOutputProcessor

A ModelOutputProcessor feladata a modell outputból adatkinyerés, majd azt a gold standarddal összevetve statisztikák készítése.

4.2. Fejlesztés

4.2.1. Globális scope

`GlobalMain.py`: ebben a szkriptben található a fő belépési pont, amely egyszerre mind a három modult lefuttatja.

`GlobalData` mappa: Ez 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-egy külön.

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

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

`.out` fájlok: az output fájlok egységes kiterjesztése. Általában "append", azaz hozzáfűző módban írat a program beléjük, hogy így meg legyenek őrizve a korábbi futtatások eredményei is.

4.2.2. A modulok szerkezete

Közös elemek

Mindhárom modul gyökérkönyvtárában megtalálható az alábbi három Python fájl.

A `config.py` nevű fájlok a konfigurációs szkriptek, 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 nyers szövegként 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 legfeljebb 2800 kérdést, igény szerint akár véletlenszerű sorrendben is. Ez a `WiC` adathalmaz `test` splitjének összes rekordjából létre képes hozni egy-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 a `randomsamples.py` szkriptem legenerál `n` darab - a `config.py`-ban meghatározott mennyiségű indexet. Ezek az indexek lesznek a kérdés-válasz párok azonosítói, én `guid`-nak neveztem el őket, amely a "globálisan egyedi azonosító" (globally unique identifier) rövidítése . Ezután megnyitjuk a fájlt, majd beolvassuk és minden egyes `guid`-adik indexű sorból eltároljuk a szót és a két mondatot

egy listában. A guid értéke mindig 0 és 1399 között van, tehát az értéke pont megegyezik a teszthalmaz hosszával, így nem történhet túlindexelés. 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.

A modul hívási verme a következő:

```
main ->
    run ->
        LabelAdder
        WordAndSentencesExtractor
        SentenceBuilder
        SentenceNormalizer
```

A négy egymás alatt található szkript egymás után hívódnak meg és mind a run szkriptnek adják vissza az irányítást a befejeztük után.

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 a nagymérteű promptom és annak speciális felépítése miatt viszont nem egy triviális feladat. Még kisebb méretű Hugging Face modellek futtatása is rendkívül körülményes. Ez egy internetelérést és rendkívül sok időt és erőforrást (CPU, GPU, RAM) igénylő művelet. Így nem meglepő, 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 ajánlott a `modelname.py` fájlban található `supported_models` lista a keretrendszerrel már tesztelt modelleket tartalmazza, ezért célszerű ezek közül választani. A `config.py` listában lévő modellek közül, annak is az első indexein elhelyezkedő modellek közül választani, mert ezek kompatibilisebbek a keretrendszerrel. A `config` fájlban lehetőség van a promptnak az utasítás részén változtatni, amely aztán a kérdések elé kerül a promptfelépítés során. Ebben lehet adni, hogy hány kérdésre válaszoljon, milyen kulcsszavakkal, terjedelemben, stb. A konzolra történő kiíratások mennyiségét is itt lehet állítani.

A futás első lépéseként a program ellenőrzi, hogy az input kérdéssor helyes-e. Ha a bemenet páratlan számú sorból áll, akkor hibás, hiszen nem egyezik meg az egyenes és fordított kérdések száma. Ilyenkor a felhasználó figyelmeztetést kap, és dönthet, hogy kilép, vagy továbblép, és a program megpróbálja helyreállítani 1 sor kitörlésével a bementetet.

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.

Ezt követően történik a 3. féltől származó könyvtárak betöltése. A szükséges fő csomagok a `torch`, `transformers`, `accelerate` és `huggingface_hub`, de bizonyos modellek további függőségeket is igényelnek. Ha valamelyik hiányzik, a Python futásidőben hibát jelez.

Ezután következik a modell-specifikus rész, hiszen a különböző modelleknek eltérő az inferencia-végrehajtó logikája. Az inferencia történhet determinisztikusan és nemdeterminisztikusan, ezt is a config fájlban lehet állítani. Az alap logika viszont minden modell esetén ugyanaz, főleg mivel a generálási stratégiám részeként minden inferencia eredménye pontosan 1 darab token. A `torch` csomag `no_grad()` kontextuskezelőjével kikapcsoljuk a gradiensek számítását a modell összes műveletére, mivel inferencia során erre nincs szükség. A program további része ebben a kontextusban fut.

Ezután történik a `transformers` csomag komponenseinek betöltése. Első lépésként a tokenizer létrehozása történik a $AutoTokenizer\ from\_pretrained()$ metódusával. A tokenizer `apply_chat_template()` metódusa a promptot a kiválasztott modell által elvárt formátumra alakítja. A modell betöltése az $AutoModelForCausalLM\ from\_pretrained()$ metódusával történik, amely — amennyiben elérhető — `safetensors` formátumú súlyokat tölt le.

A modell és a tokenizált bemenet áthelyezése a megfelelő eszközre a $model.to(device)$ és $inputs.to(device)$ hívásokkal történik. A válasz generálása a `generate()` metódussal történik, amely jelen konfigurációban pontosan egy új tokent állít elő $max\_new\_tokens = 1$. A kapott tokenhalmazt a `tokenizer.decode()` segítségével alakítom vissza szöveggé.

A futás végén megtörténik a modell válaszainak fájlba írása, továbbá a futás során keletkezett meta-információk is külön fájlokba kerülnek.

A modul hívási verme a következő:

```
main ->
    run ->
        TorchApiHandler ->
            TransformersApiHandler ->
                Builder
```

Mindegyik szkript rekurzívan meghívja a nyíllal tőle jobbra jelzett szkriptet, majd azok mindegyikének befejezte után veszi vissza az irányítást.

Ha sikeresen lefutott a válaszgenerálás, akkor a kérdések hosszával megegyező számú választ kaptunk. Ellenkező esetben sajnos meg kell ismételni a generálást, hiszen nem triviális, hogy melyik válasz melyik kérdéshez tartozott - kivéve ha kérdés-válasz párokként mentettük el az eredményt. Azonban az eredményként kapott modell válaszok még helyes válaszgenerálás esetén is csak egy halom "Yes" és "No", mindegyik új sorban, amelyből önmagában nehéz statisztikai következtetéseket levonni. Éppen ezért van szükség a ModelOutputProcessor modulra, amely érthetővé teszi az eredményeket.

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.

A modul, és így az egész program végeredményeként az alábbi statisztikákat számolom ki. Bemutatom a számolási módszer képletét és helyességét is.

Informatikusként természetesen szeretnénk az eredményeket számszerűsíteni. Különösen igaz ez a `ModelOutputProcessor`, esetén, hiszen az eredmények könnyen számszerűsíthetők. Ez talán a legkönnyedebb módja a különböző futtatások összehasonlításának és ez teszi lehetővé a következtetések levonását. Éppen ezért minden eredményt egy 0 és 100% közé normalizált számként fejez ki a programom. Az eredményként az alábbi statisztikai indikátor mérőszámokat számítja ki a modul:

- Konzisztens válaszok aránya
- Pontos válaszok aránya
- Konzisztensen pontos válaszok aránya
- Bizonytalan válaszok aránya
- Konzisztensen bizonytalan válaszok aránya

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, függetlenül attól, hogy az egyenes vagy fordított. Ezért egy százalékban kifejezett mérőszámban eltároltam az egész adathalmazra levetített pontosság mértékét is. Pl. 50% esetén a válaszok 50%-a egyezik meg a gold standardbeli válasszal.

Konzisztensen pontosság

A konzisztensen pontos válaszok egyszerűen azok, amelyek konzisztensek és pontosak is. Magyarul azon túl, hogy az egyenes és a fordított kérdésre adott "igen" vagy "nem" válasz(ként generált token) megegyezik, mindez a gold standard válasszal is megegyezik. Emiatt ha a válasz nem "igen" vagy "nem", az automatikusan nem konzisztensen pontos.

Bizonytalan

Ha egy egyenes-fordított kérdéspár **legalább egyikére** a válasz se nem "igen", se nem "nem", akkor azt a sort "bizonytalannak" számítom.

Konzisztensen bizonytalan

Ha egy egyenes-fordított kérdéspár **mindkét kérdésére** a válasz se nem "igen", se nem "nem", akkor azt a sort "konzisztensen bizonytalannak" számítom.

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, és a konzisztens pontosság .

4.3. Indexer

Ha megkaptuk az 5 értéket (Konzisztens, Pontos, Konzisztensen pontos, Bizonytalan, Konzisztensen bizonytalan) a megfelelő tartományban, akkor már nincs más dolgunk, mint ellenőrizni az alábbiakat: Az adathalmaz sor-kérdés-egyenes válasz-fordított válasz-gold standard sor guid-jai **pontosan megegyeznek-e**. A guid az eredeti, WiC adathalmazban elfoglalt sorszám, amelyre indexként tekintünk a beolvasás után. A guid az indices.out fájlban található Ilyen nagy mennyiségű adat esetén komoly problémát jelent a manuális indexelés, kiváltképpen mivel a kérdések sorrendje eleve véletlenszerű. Ráadásul indexer nélkül körülményes a ModelOutputProcessor eredményinek validálása. Éppen ezért a ModelOutputProcessor modulban létrehoztam egy indexer függvényt, amely az az egyenes kérdést, a fordított kérdést, az egyenes kérdésre adott választ, a fordított kérdésre adott választ, a gold standard választ, valamint a ModelInputPreparerben létrehozott kérdés `guid`-jait indexeli, kapcsolja össze. Így visszakereshetővé vált, hogy melyik választ melyik kérdésre adta a modell, és hogy az megegyezik-e a gold standarddal.

4.4. Tesztelés

4.4.1. A programok futtatása

A PyCharm fejlesztői környezet 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.

4.5. Egyebek

4.5.1. Elnevezési stílus

Igyekeztem a Clean Code [18] elveit követve a változók, függvények, osztályok és metódusok szerepét sokatmondó, egyértelmű, könnyen kereshető és refaktorálható, leíró nevekkel ellátni. A fájlok és osztályok neveit egységesen *PascalCase* stílusban adtam meg, minden egyéb változót pedig *camelCase* stílusban.

4.5.2. Hibakezelés

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

4.5.3. Dupla importok

A moduljaimban szinte minden fájlban észrevehető, hogy az importok egy try-catch blokkban vannak megadva. Ez duplikált importálásnak tűnhet, ám nem az, hiszen a verzérlési szerkezet biztosítja, hogy ha az első elérési út megtalálása meghiúsul, akkor a másik útvonalról töltődjenek be a szükséges modulok. Erre azért volt szükség, mert mindenképpen el szerettem volna érni, hogy a projekt gyökérkönyvtárából és modul-szinten is futtathatóak legyenek a programok.

4.5.4. Kommentek

A kommentek terén törekedtem, hogy minden angolul legyen, ám ezt egy idő után feladtam, és az újabb részek kommentjeit magyarul írtam. Mivel a céljaim a helyes funkcionalitás és a karbantarthatóság voltak, ezért bevallom, elég sok kikommentezett kódrészlet és számos nyúlfarknyi megjegyzés található, ám ezek az átláthatósághoz és a jövőbeli fejlesztésekhez szükségesek.

4.5.5. 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.

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

4.5.6. 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 jelentősebben újra kellene írni a logikáját. Az inputok beolvasása, modell választás és az outputok feldolgozása során is 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.

4.6. Jövőbeli tervek

Hosszútávú célom egyszer közzétenni a keretrendszert. Ehhez azonban még sok fejlesztési "mérföldkövet" kellene elérnem. Egy letisztult webes, vagy konzolos CLI felület sokat segítene, hogy a projekt szélesebb körben, több ember számára elérhető lehessen. A Hugging Face Hub [19] alkalmas platform lehet erre. Továbbá egy adatbázis a jelenlegi nyers szöveg-alapú adatkezelés helyett nagy előrelépés lenne.

5. fejezet

Kísérlet

A végső cél, amiért az egész szoftver keretrendszer készült, az volt, hogy legalább 2 modellt összehasonlíthassunk minél nagyobb mennyiségű kérdésen.

5.1. Két konkrét modell kiértékelése véletlenszerű mintahalmazon a keretrendszerben

Ahhoz, hogy módszertanilag helyes legyen és látszódjon, hogy az eredmények nincsenek beégetve a keretrendszerbe, a Témavezetőmmel megegyezve a kérdéseket random index generátor függvény segítségével véletlenszerűen választottam ki a WiC-adathalmaz bejegyzéseiből. A kérdések között nincs ismétlődés és minden modell ugyanazt a kérdéskészletet kapja meg feladatként. A kérdések 100, véletlenszerűen kiválasztott WiC adathalmazbeli mintából származnak.

5.2. Nehézségek

Akkor beszélhetünk egyáltalán konzisztenciáról, azt tekinthetjük a konzisztens viselkedés első szintjének, ha a modell *reagál a bemenetre*, azaz a válaszának egyáltalán valami köze van a feltett kérdésekhez. A legnagyobb nehézséget a dolgozatomban egyértelműen a nem megfelelő formátumú és/vagy mennyiségű kimenet jelentette a modell válaszadás során, amely miatt gyakran az egészet meg kellett ismételni. Ugyanis ha nincs reláció a modell válasza és az egyes kérdések között, nem lehet egyértelműen megfeleltetni a válaszokat a kérdésekhez, így a kiértékelés sem lehetséges. Ekkor az összes mérőszám 0.

A vizsgált nyelvi modellek jelentős része nem képes stabil és konzisztens döntést hozni a WiC-feladat esetén. Ha a maximum output tokenek mennyisége nincs lekorlátozva 1-re, gyakran irreleváns generatív válaszokba esnek (pl. “Hello, how can I help you today?”), vagy repetitív tokenflooding jelenséget mutatnak, egy nagy valószínűségű tokenre “ragadva rá”, majd azt ismétli determinisztikusan, magas temperature nélkül (“Yes Yes Yes…” vagy “No No No…”). Ez a *mode collapse* egy tipikus megnyilvánulása, mely azt mutatja, hogy a modellek nem értik megfelelően az instrukciót, nem tartják a formátumot, és egyenes-fordított párok esetén extrém alacsony konzisztenciát adnak. Ezen jelentősen segített a `max_new_tokens` paraméter 1-re állítása a `transformers` könyvtár

`_BaseModelWithGenerate` típusának a `generate` metódusában. Ez minden esetben garantálta, hogy a modell pontosan 1 tokenben, 1 szóban válaszoljon.

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

A modelleket a Témavezetőm ajánlásával együtt választottam ki. Ő ajánlotta, hogy a LMArena [3] és a Hugging Face [20] platformokon lévő modelleket vizsgáljam meg, de a konkrét modell példány kiválasztását rám bízta. 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.

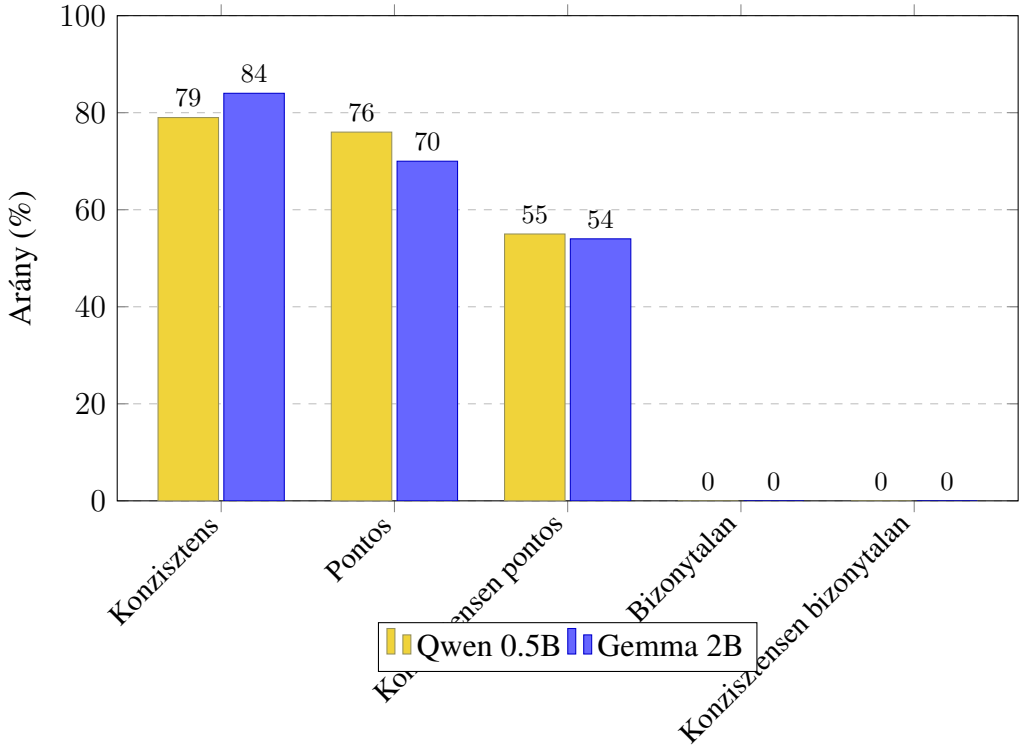
Így a keretrendszeremet és a Colab jegyzetfüzetemet elsősorban a

- Microsoft/Phi-4-mini-instruct [21],
- Google/Gemma-2-2b-it [22],
- Qwen1.5-1.8B-Chat [23],
- Qwen/Qwen2.5-1.5B-Instruct [24] és a
- Qwen/Qwen2.5-0.5B-Instruct [24]

modelleken teszteltem, melyek paraméterszáma 0.5 és 4 milliárd közötti. A méretükhöz képest gazdag szókinccsel és jó nyelvérzékkel rendelkeznek. Hangsúlyozom, hogy nem a gemma-2b-it, hanem a gemma-*2*-2b-it változatot választottam.

5.4. 2 modell összehasonlítása 100 véletlenszerű kérdésen

Végül a `Qwen/Qwen2.5-0.5B-Instruct` és a `Google/Gemma-2-2b-it` modelleket hasonlítottam össze. Az előbbi az Alibaba Cloud gyártmánya, míg utóbbi a Google szüleménye. A választással megfigyelhetjük 2 eltérő méretű (0.5B vs 2B) és gyártójú, ezáltal architektúrájukban is némileg különböző *instruct* modell különbségét. Keretrendszerem érdeme, hogy végül sikerült a 2 modell hibátlan válaszadását 100 véletlenszerű kérdésre (és a fordított párjukra) elérni. A modellek hibátlanul csak "Yes" és "No" válaszokat adtak. Az alábbi ábrában `n` modell összehasonlítása látható `m` véletlenszerűen kiválasztott kérdésen.



5.1. ábra. Két modell kiértékelése 100-100 véletlenszerű kérdésen...

5.5. Google Colab futtatás

A középső modul egyszerűsített változata elérhető Google Colab-on, a mellékletben található linken. 6

Mivel Google Colabot használtam, azon belül is GPU-s futtatókörnyezetet, így a kiértékelés megfelelően gyors volt. A webes környezet és a kikapcsolt valószínűségi változók és az inferenciák egyesével történő végigvárása miatt elmondható, hogy a kiértékelés módja:

- reprodukálható, azaz bárki meg tudja ismételni a kísérletet,
- determinisztikus, tehát ugyanarra az inputra mindig ugyanaz lesz az output, nem függ a véletlentől,
- és izolált, azaz a modell egyik futtatásban sem "tud" semmit a korábbi bemeneteiről és válaszairól.

Ennek ellenére a Google Colab környezet nem egy hatékony módja a kiértékelésnek a korábban felsorolt gyengeségei miatt, helyette a lokális PyCharm keretrendszerem biztosít egy kényelmesebb és sokkal átláthatóbb környezetet.

6. fejezet

Konklúzió

Tisztán látható, hogy a Qwen2.5-0.5B-Instruct modell előnyben részesíti a "nem" választ, amelynek fő oka valószínűleg az, hogy nincs elég lexikális tudása, hogy ezeknek a WiC adathalmazbeli ritka szavaknak a pontos jelentését emberi szinten modellezni tudja, így a megfelelő tudás hiányában a biztonságosabb "nem ugyanazt jelentik" predikcióba esik vissza. Ezt az egyoldalúságot valószínűleg továbberősíti az adathalmaz mondatainak jellemző nem-köznyelvi jellege, hiszen nagyrészt régies, tudományos, irodalmi szövegeket tartalmaz. Ilyesfajta szövegeket egy ilyen kisméretű, köznyelvi angol megértésére szánt modell nem tud megfelelően ábrázolni és értelmezni.

Említésre méltó az inferenciák időkülönbsége is a két modell között, amelyben viszont jelentősen nyert a Qwen. A Qwen modell esetén átlagosan 6 másodpercet vett igénybe egy kérdés megválaszolása, míg a Gemma modell esetén átlagosan kb. 35(!) másodperc volt.

Habár nem sikerült olyan mértékben automatizálnom a keretrendszert, mint ahogy eredetileg megálmodtam, az elkészült keretrendszer jelentősen megkönnyítette a végső kísérlet elvégzését és így a saját munkámat. Részben így is manuálisan kellett feldolgozni az adatokat, de a munkafolyamatok túlnyomó többségét sikerült teljesen automatizálnom. Ha az elkészült munkámmal legalább egy embernek segítettem, akkor már megérte. Bízom benne, hogy a keretrendszerem és a kísérletem hasznos lesz a jövőben mások számára is.

Irodalomjegyzék

[1] Jingyuan Yang és tsai. „Enhancing Semantic Consistency of Large Language Models through Model Editing: An Interpretability-Oriented Approach”. *Findings of the Association for Computational Linguistics: ACL 2024*. Szerk. Lun-Wei Ku, Andre Martins és Vivek Srikumar. Utolsó megtekintés: 2025-12-10. Bangkok, Thailand: Association for Computational Linguistics, 2024. aug., 3343–3353. old. DOI: 10.18653/v1/2024.findings-acl.199. URL: https://aclanthology.org/2024.findings-acl.199/.

[2] Hugging Face. *Hugging Face Open LLM Leaderboard Model Comparator*. https://huggingface.co/spaces/open-llm-leaderboard/comparator. Utolsó megtekintés: 2025-12-10.

[3] Wei-Lin Chiang és tsai. *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*. Utolsó megtekintés: 2025-12-10. 2024. arXiv: 2403.04132 [cs.AI]. URL: https://arxiv.org/abs/2403.04132.

[4] Ollama Team. *Ollama: Run LLMs locally*. https://ollama.com. Utolsó megtekintés: 2025-12-10. 2024.

[5] Georgi Gerganov. *llama.cpp: Port of Meta's LLaMA models in C/C++*. https://github.com/ggml-org/llama.cpp. Utolsó megtekintés: 2025-12-10. 2024.

[6] Woosuk Kwon és tsai. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. https://github.com/vllm-project/vllm. Utolsó megtekintés: 2025-12-10. 2023.

[7] LM Studio Team. *LM Studio: Desktop application for running local LLMs*. https://lmstudio.ai. Utolsó megtekintés: 2025-12-10. 2024.

[8] oobabooga. *Text Generation WebUI*. https://github.com/oobabooga/text-generation-webui. Utolsó megtekintés: 2025-12-10. 2024.

[9] GPT4All Team. *GPT4All: Local large language model ecosystem*. https://gpt4all.io. Utolsó megtekintés: 2025-12-10. 2024.

[10] 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)*. Utolsó megtekintés: 2025-12-10. New York, NY, USA: ACM, 1986, 24–26. old. URL: https://dl.acm.org/doi/10.1145/318723.318728.

[11] Paul-Edouard Sarlin és tsai. *SuperGlue: Learning Feature Matching with Graph Neural Networks*. Utolsó megtekintés: 2025-12-10. 2020. arXiv: 1911.11763 [cs.CV]. URL: https://arxiv.org/abs/1911.11763.

[12] Alex Wang és tsai. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*. Utolsó megtekintés: 2025-12-10. 2020. arXiv: 1905.00537 [cs.CL]. URL: https://arxiv.org/abs/1905.00537.

[13] Mohammad Taher Pilehvar és Jose Camacho-Collados. „WiC: the Word-in-Context Dataset for Evaluating Context-Sensitive Meaning Representations". *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Utolsó megtekintés: 2025-12-10. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. jún., 1267–1273. old. DOI: 10.18653/v1/N19-1128. URL: https://aclanthology.org/N19-1128/.

[14] George A. Miller. „WordNet: A Lexical Database for English". (1994). Utolsó megtekintés: 2025-12-10. URL: https://aclanthology.org/H94-1111/.

[15] Ari Holtzman és tsai. „The Curious Case of Neural Text Degeneration". *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. Utolsó megtekintés: 2025-12-10. 2020. arXiv: 1904.09751. URL: https://arxiv.org/abs/1904.09751.

[16] OpenAI Community Forum. *What is the context window of GPT-4?* https://community.openai.com/t/what-is-the-context-window-of-gpt-4/701256. Utolsó megtekintés: 2025-12-10.

[17] Gábor Berend. *11_phi.ipynb*. https://colab.research.google.com/drive/1GQRiTDNWwNPP_PPARYd1swY1Oiai9Ey_. Utolsó megtekintés: 2025-12-10. 2025.

[18] 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.

[19] Hugging Face. *Hugging Face Hub*. https://huggingface.co/. Utolsó megtekintés: 2025-12-10. 2025.

[20] Thomas Wolf és tsai. „HuggingFace's Transformers: State-of-the-art Natural Language Processing". *arXiv preprint arXiv:1910.03771* (2019). Utolsó megtekintés: 2025-12-10. URL: https://arxiv.org/abs/1910.03771.

[21] Microsoft. *Phi-4-mini-instruct*. https://huggingface.co/microsoft/Phi-4-mini-instruct. Utolsó megtekintés: 2025-12-10. 2024.

[22] Google. *Gemma-2-2b-it*. https://huggingface.co/google/gemma-2-2b-it. Utolsó megtekintés: 2025-12-10. 2024.

[23] Qwen. *Qwen1.5-1.8B-Chat*. https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat. Utolsó megtekintés: 2025-12-10. 2024.

[24] Qwen Team. *Qwen2.5: A Party of Foundation Models*. Utolsó megtekintés: 2025-12-10. 2024. szept. URL: https://qwenlm.github.io/blog/qwen2.5/.

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 11.

. . . . . .
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. Külön köszönetet szeretnék mondani mindazoknak akik tesztelték és átnézték a munkámat.

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.