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
Programtervez informatikus szakos hallgató

Témavezetõ:
Dr. Berend Gábor
egyetemi docens

Szeged
2025

Feladatkiírás

A hallgató feladata egy olyan keretrendszer megvalósítása, amely lehetvé 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 nev feladat megoldása során az egyes inputokban szerepl mondatpárosok sorrendjének megcserélésére.

Tartalmi összefoglaló

A téma megnevezése

A nagy nyelvi modellek szemantikus képességeinek konzisztenciájának vizsgálata.

A feladat megfogalmazása

A nyelvi modellek gyakran érzékenyek a felhasználótól kapott prompt szósorrendjére, emiatt nincsen garancia arra, hogy szemantikailag azonos értelm, de más szósorrend input promptra konzisztens viselkedést mutatnak. 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 szósorrend 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 futtatásával a kiválasztott modell a kért mennyiség, egyértelm igen/nem választ képes generálni az eldöntend kérdésekre. A válaszainak helyessége másodlagos, de nem elhanyagolható. A nyelvi modelleket a Hugging Face platformról fogom választani.

A program bemenete tetszleges sornyi bejegyzés a Word-in-Context adathalmaz `test` splitjébl (vagy tetszleges, ezzel megegyez formátumú kérdéshalmazból), és tetszleges modell a Hugging Face platformról.

A megoldás kulcsa olyan szoftver fejlesztése, amely mindezt minél egyszerbbé, 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 [**martin2008cleancode**] alapján

Elért eredmények

Elkészült a keretrendszer, egy Python konzolos program, amely képes tetszleges 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 eredményekbl.

Kulcsszavak

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

Tartalomjegyzék

Motiváció

Az informatika és a nyelvtechnológia fejldésével a generatív mesterséges intelligencia mindennapjaink részévé vált. A nyelvi modellek kiértékelésére számos teljesítményteszt (benchmark) fejldött ki az évek során. Az egyik legnépszerbb ilyen teljesítményteszt a SuperGLUE Benchmark, amely 8, emberi nyelvi megértést igényl nagyon nehéz érvelés-orientált feladat elé állítja a nyelvi modelleket. A WiC (Word-in-Context) probléma a SuperGlue 8 feladatának egyike. Az emberi szöveget ért nyelvi modellek fejlesztése kiemelten fontos mind az akadémiai kutatásban, mind a gyakorlati alkalmazásokban. Az angol nyelv 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 elnyt jelent mind a munkahelyeken, mind a magánéletben. IT területen különösen nagy elnyt jelent az AI megfelel 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, fleg a nyelvészeti informatikai területen dolgozók számára.

1. fejezet

Elméleti háttér

1.1. Problémafelvetés

Architektúrájukból adódóan a modellek esetén nincs garancia, hogy bizonyos emberek számára triviális invarianciákat konzisztensen kezeljenek, erre fókuszál a szakdolgozatom. Hogy majd az eredményekbl következtetést lehessen levonni, két hipotézist fogalmazok meg. Egyrészt valószínsíthet és feltételezem, hogy minél nagyobb paraméterszámú egy model, annál jobban fog teljesíteni a megfogalmazott feladatra. Továbbá méretnövekedéssel történ teljesítményjavulás mértéke is kérdéses és érdekel. Másrészt azonos méret 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 különböz cégek által gyártott modellekre. Emiatt azt is feltételezem, hogy lesznek különbségek az ilyen azonos méret, de eltér architektúrájú modellek között. A szakdolgozatom, pontosabban a szoftverem tesztelésének eredményei ezeket a hipotéziseket vagy megersítik, vagy megcáfolják és ezekre a feltevésekre majd a konklúzió fejezetben reagálok.

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

1.2. Hasonló megoldások

Az elmúlt években számtalan, a nagy nyelvi modellek szemantikus konzisztenciájával kapcsolatos cikk jelent meg, jól mutatva, hogy a téma nagy népszerségnek örvend [**yang-etal-2024-enhancing**]. A törekvés, hogy generatív modellek lokális telepítésére, futtatására és összehasonlítására szolgáló eszközöket hozzunk létre nem újkelet, hiszen egy jó modell jelentsen megkönnyíti a munkafolyamatokat, a lokalizálása pedig biztonságosabbá és olcsóbbá teszi a használatukat. Számos hasonló témájú és ötlet projekt született az elmúlt években. Az alábbiakban bemutatok párat.

1.2.1. Nyelvi modellek összehasonlítását segít eszközök

A különböz nyelvi és nem nyelvi modellek összehasonlítására egyre több projekt létezik, amelyeket nagy érdekldés övez és nagy aktív felhasználó és fejleszt-bázissal rendelkeznek. Például a Hugging Face Open LLM Leaderboard Model Comparator [**hf_llm_comparator**],

amely elssorban nyílt forráskódú ingyenes modellek összehasonlítására alkalmas, akár webesen, akár saját eszközre telepítve, továbbá a LMArena [**chiang2024chatbot**] webes felület, ahol fizets modellek is kipróbálhatók korlátozottan, viszont csak a weboldalon keresztül. Ezeken a platformokon különböz modellek teljesítményét lehet összehasonlítani különböz 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 eltt átelemeztem az jelenlegi legjobb módszereket és nyílt forráskódú nyelvi modelleket a Témavezetm 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 segít eszközök

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

- A Témavezetm által ajánlott Ollama [**ollama**] CLI + desktop alkalmazás helyi API-jal sok nyílt modellt támogat.
- A szintén a Témavezetm által ajánlott llama.cpp [**llama_cpp**] egy nyílt forráskódú C/C++ projekt.
- A szintén a Témavezetm által ajánlott `vllm` [**kwon2023efficient**] szintén egy aktívan fejlesztett nyílt forráskódú projekt, amely elssorban a memóriagazdálkodásra fókuszál.
- Az `LM Studio` [**lmstudio**] grafikus felület. Modellmenedzsment, többszörös modellváltás, helyi inferencia elérhet benne.
- text-generation-webui [**textgen_webui**]: webes frontend, backendként pl. `llama.cpp`, nagyon rugalmas, sokféle modellt és konfigurációt támogat.
- A `GPT4All` [**gpt4all**]: kezdeti belép a helyi LLM-használatba CPU-barát, alacsony küszöb, egyszer használat jellemzi.

1.3. Az én szoftverem elnyei korábbiakhoz képest

A fent említett platformokhoz hasonlóan a megoldásom lehetvé 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 elbbiek, 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 tetszlegesen sok inferenciát kiszámíttathatunk, - olyan, mintha

mindegyik prompt "új chatbe" kerülne - amely jelents elny 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 elérhet és nyílt forráskódú eszközöket (Python, GitHub, Hugging Face) használtam.

1.4. A többértelmség problémája a természetes nyelvekben

A természetes nyelvekben a programozási nyelvekkel ellentétben egy szónak több, egymástól teljesen elkülönül jelentése is lehet. Például az "egér" szó jelenthet egy számítógépes perifériát vagy egy állatot, és a helyes értelmezéshez a környez szavakat ismerni kell. Az ilyen jelleg többértelmségek 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 feladatot 2019-ben fogalmazták meg Mohammad Tahmed Pilehvar és munkatársai, abból a célból, hogy különböz transzformer és szóbeágyazasos modelleket vizsgáljanak a feladat által megfogalmazott teljesítményteszten. A WiC feladat lényege, hogy egy adott szó két különböz mondatbeli elfordulásáról eldöntse, hogy azonos értelemben szerepel-e. Az ezt a feladatot megfogalmazó adathalmazt alkalmasnak találtuk a Témavezetmmel az általa megfogalmazott teljesítményteszt, a nagy nyelvi modellek szemantikus képességei konzisztenciájának vizsgálata elvégzéséhez. A kérdések már eleve csoportosítva vannak azonos és különböz jelentés mondatpárokként.

A Word-in-Context háttere

A WiC csapata alapveten a folyamatosan fejld modelleknek igyekezett egy nehezebb, korszer 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 tervezésükbl adódóan nem képesek modellezni a szavak szemantikájának dinamikus természetét, vagyis azt a tulajdonságot, hogy a szavak potenciálisan különböz 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 jellemzen neurális hálókat használnak a modellek szójelentés-értelmez képességeinek fejlesztésére.

1.6. Egy rendszer feladata a WiC adathalmazon

Amikor valaki kiértékel rendszert fejleszt a WiC benchmarkra, annak feladata a szavak szándékozott jelentésének azonosítása. A WiC egy bináris osztályozási feladatként van megfogalmazva. Adott egy többjelentés szó, amely mindkét mondatban elfordul, továbbá egy szófaj címke, kett 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 fnév lehet. A célszóhoz két eltér szövegkörnyezet tartozik. Ezen szövegkörnyezetek mindegyike a w egy specifikus jelentését váltja ki. A feladat annak megállapítása, hogy a w elfordulásai a két szövegkörnyezetben ugyanannak a jelentésnek felelnek-e meg, vagy sem. Tehát a célszó ugyanazt a jelentést hordozza-e két különböz szövegkörnyezetben, vagy eltért. Ez egy összetett NLP probléma, mivel ötvözi a szójelentés-egyértelmsítés (Word Sense Disambiguation, WSD) és a kontextuális beágyazások elemeit, így a szójelentés-egyértelmsítés végrehajtásaként is értelmezhet.

1.7. Az én céljaim ehhez képest

Ez a SuperGlue feladat, a Word-in-Context alapján készíthetek olyan promptok, amelyek hasznosnak bizonyulhatnak az inkonzisztens viselkedés detektálására. A kutatásom célja a témámnak megfelelen a modellek teljesítményének összehasonlítása a WiC-bl 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`?
```

kérdésekre mindig ugyanazt kellene, hogy válaszolják (változatlan w, $s1$ és $s2$ esetén, ahol w egy szó, $s1$ az els példamondat, és $s2$ a második példamondat). Ezek az invarianciák - a két mondat sorrendjének felcserélése - az ember számára könnyen felismerhetek, így az emberi válaszadásra nincsenek befolyással, ám a nyelvi modelleket, fleg az alacsony, "csak" pár milliárd paraméterszámúakat könnyen összezavarja.

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 sorrend kérdések száma is **minden esetben** szigorúan megegyezik. Ha

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

mégsem teljesül a fentiek valamelyike az inputra, 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 különböz mondatbeli elfordulása 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 [**lesk1986automatic**] egy klasszikus szóértelem-felismer módszer, amely a szótári definíciók és a kontextus összevetésével próbálja meghatározni a szó legmegfelelbb 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 Word-in-Context adathalmaz egy jó minség, nyelvészeti szakértk által készített benchmark adathalmaz. A mondatok a WordNetbl, a VerbNetbl és a Wikiszótárból származnak. A WiC adathalmaz segítségével megvizsgálhatjuk, hogy egy rendszer (modell, algoritmus) mennyire képes a szavak jelentését megérteni különböz kontextusokban. A WiC feladat része a SuperGlue [**sarlin2020superglue**] 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 [**wang2020superglue**]

A SuperGLUE a GLUE továbbfejlesztett változata, amelyet 2019-ben azért vezettek be, mert az elz versenykörnyezet, a GLUE feladatai könnyvé váltak a modern, nagy-

teljesítmény NLP-modellek számára. [**wang2020superglue**] 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 egyszeren felhasználható. Ezt segíti el 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 jellemzen 3 részre vágják: tanító, validációs és teszthalmazra. Ennek az aránya eltér lehet, de a tanítónak jóval nagyobbnak kell lennie, mint a másik kettnek, és a teszthalmaznak nagyobbnak kell lennie, mint a validációs halmaznak. 80-10-10%-os eloszlás a standard, ezzel, vagy hasonló eredménnyel érhetek 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 végs 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**]

1.9.1. 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, küls erforrások használatával. Ez az eredmény megközelíti a kézi, emberi szint kiértékelést, melynek a fels határa 80% körüli. A kézi kiértékelésnek és a SenseBERT megoldásának egyszersített változatát én is elvégeztem. A kézi kiértékelésemmel közel 65, míg egy egyszer 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 (véletlenszer 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 mennyiség szöveges adat feldolgozásával és mélytanulási technikák alkalmazásával sajátítják el a nyelv megértését és elállítását. Az LLM-ek, mint például az OpenAI GPT-sorozata, a Google Gemini vagy a Meta LLaMA modelljei, különböz architektúrákat használnak, de leggyakrabban a transzformer alapú megközelítést alkalmazzák. Mint nyelvi modellek, az LLM-ek úgy sajátítják el ezeket a képességeket, hogy óriási mennyiség szövegbl, 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 jellemzen 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 modelltl és a paraméterezéstl. 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 (elre meghatározható) és részben véletlenszer is. A prompt jellemzen szöveges formájú **nyelvi modellek esetén**, mint ahogy a válasz is, de

egyre elterjedtebbek a hangalapú bemeneti és kimeneti formák is. Egy jelents elnye a promptolásnak, hogy nem csak egy elre 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 egyszerség kedvéért általában "a modell futattásaként" fogok hivatkozni.

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

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

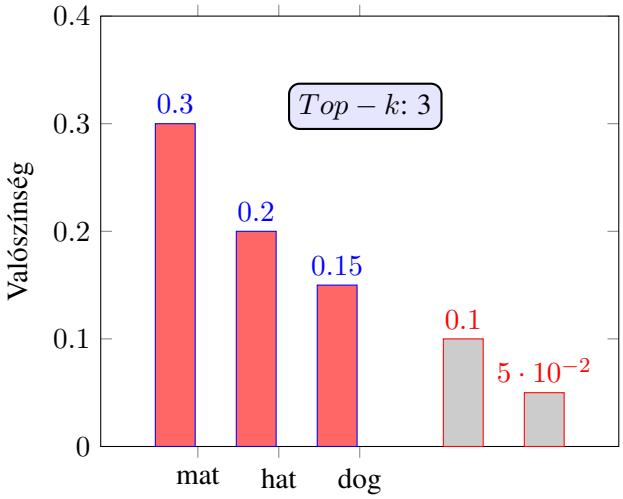
A modellek teljesítményének vizsgálata könnyebb lehet, ha a futtatás determinisztikus. Azt jelenti, hogy a modell mindig ugyanazt a kimenetet adja ugyanazon bemenet esetén, mivel a véletlenszerséget kizáró paraméterekkel (pl. $temperature = 0.0$, $top - k = 0$, $top - p = 1.0$) mködik. A hmérséklet (temperature) a nyelvi modellek válaszainak kreativitását szabályozza (magas értéknél változatosabb, alacsonynál determinisztikusabb kimenet), a top-k pedig korlátozza a következ szó választékát a k legvalószínbb tokenre, míg a top-p (más néven nucleus sampling) dinamikusan választja ki a valószínségi eloszlás egy részhalmazát (pl. a legvalószínbb tokeneket, amelyek összege eléri a p küszöböt) [**holtzman2020curious**].

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

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

A modell célja, hogy megtalálja a legmegfelelbb szót a hiányzó helyre. Ehhez különböz mintavételi stratégiákat alkalmazhat, például a top-k vagy top-p eljárást. A top-k módszer során a modell kiszámítja az összes lehetséges folytatás valószínségét, majd ezek közül kiválasztja a k legvalószínbbet, a többit pedig elveti. A kiválasztott k szóból ezt követen véletlenszeren 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 legvalószínbb, csökken eséllyel: mat, hat, dog, sofa, floor.



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

A $top-p$ mködése hasonló. Annyiban tér el, hogy nem egy fix számú legvalószínbb szót választunk ki, hanem az odaill szavak valószínsége szerint. A p egy 0.00-tól 1.00-ig terjed valós szám. A szavakat akkumulatívan vesszük számításba, amíg a p értéke nagyobb, mint a soron következ legvalószínbb szó valószínsége, addig belevesszük a szót és ugrunk elre. Ellenkez 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 vehet 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. Ötletem megvalósításához a Témavezetm által készített és javasolt Google Colab jegyzetfüzetet [**berend2025phi**] fejlesztettem tovább. Ez egy egyszer, de hatékony jegyzetfüzet, amellyel egy tetszleges modell válaszadásra bírható. A környezettel egy darabig, a modell válaszok legeneráltatása erejéig effektíven és eredményesen tudtam a kiértékel keretrendszert fejleszteni. Az elkészült jegyzetfüzet 6 jelenlegi formájában képes a meghatárzott számú, akár mind a 2800 Word-in-Context-alapú kérdésre választ generáltatni a meghatározott modellen, így rendelkezik a keretrendszer középs moduljának 4.1 funkcionalitásával. Használata egyszer: csak le kell futtatni sorban az összes cellát, mely akár egyetlen kattintással is elvégezhet A Google Colab egyszer 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 id a távoli hardverekhez, limitált erforrás-hozzáférés, perzisztens adattárolási lehetségek hiánya és egyéb korlátozások, amelyek csak elfizetéssel kerülhetek ki. Emiatt PyCharm fejleszti környezetet használtam a keretendszer elkészítésére.

4. fejezet

A szoftver: modellt futtató és kiértékel 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 elérhet a kitzött elemek között. A teljes forráskód az összes korábbi verzióval együtt a
https://github.com/Fabbernat/Thesis GitHub repozitóriumban található.

A projekt els verziója kritikákat kapott, ugyanis nehezen átlátható és nagyban AI generált, rossz minség kódra alapszik, így 2025 nyarán újrakezdtem a fejlesztést. Az új szoftvernek 2 f fejlesztési szempontja volt.

Az egyik, hogy mind a 3 részfeladat egy kattintással elvégezhet legyen a Clean Code [**martin2008cleancode**] módszereivel, tehát nagyon egyszeren futtatható legyen. Ez a 3 részfeladat:

- A WiC adathalmazból (vagy azzal megegyez 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 egyszeren cserélhetek 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 mellzhet a projekt megértéséhez. A `src/Framework` 3 almodult tartalmaz, amelyek mindegyike egy önálló, de nagyon egyszeren futtatható programot tesz ki. Egyesítve is lehet futtatni, de több indoka is van, hogy miért inkább egyesével érdemes elindítani a programokat. Az els és utolsó modul offline futtatható és nem erforrásigényes, a hibák esélye minimális. A középs modul a f

ok, amelyben rengeteg a hibaforrás és a nemdeterminizmus. Online API-hívásokat végez és GPU-t igényel (GPU hiányában pedig hatalmas erforrásigényt és számítási kapacitást), ráadásul sok harmadik féltl 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 mködött, csak úgy "volt". A program egyébként is csak a modell inputjainak elkészítését tudta, a modell inferenciát és a válasz feldolgozást már manuálisan kellett elvégezni. Csak a mostani `ModelInputPreparernek` megegyez rész funkcionalitásával rendelkezett standard Python környezetben. A Hugging Face modell inferenciájára (a mostani `HuggingFaceModelInferencer` 4.1 modullal megegyez 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 els 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 lév 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 cím könyv elolvasása hozta, amely hatására átértékeltem a programozói tudásomat és a szakdolgozatomat. Láttam, hogy modulárisan, objektumorientált módszerrel és az egymásra halmozódó függségek 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 paraméterezhet. Én általában csak "modulokként" hivatkozom a keretrendszerre, hiszen három, egymástól jól elkülönül feladatú és funkciójú modulból áll. Így az egyes modulok egyedülállóan is futtathatóak, akkor is, ha a másik kett nem létezik, vagy meghibásodik. Mindegyik modul egy f futtatható állományt tartalmaz, egy-egy `main.py` fájt egy-egy `main` belépési ponttal.

Felhívom a figyelmet arra, hogy mivel a szoftvert idnként módosul, így a legújabb GitHub verzió konfigurációi, paraméterei, mködése stb. részben eltérhetnek a leírttól. A leírás a 2025 év végi verzióra vonatkozik és a késbbi verziókban kisebb eltérések elfordulhatnak. A legtöbb eltérés korábbi verziók visszaállításával, vagy a config és az input fájlok módosításával kiküszöbölhet.

ModelInputPreparer

A `ModelInputPreparer` modul inputjai 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 `Word-in-Context` adathalmaz `test` splitjének rekordjaiból hozza létre a kérdéseket, de saját, megegyez formátumú kérdésekre is mködik. 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 történik a távoli Hugging Face modell letöltése, telepítése és futtatása. Feladata a modell futtatása az elkészített prompton, válasz generálása.

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 f 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 ebbl 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

4.2.2. A modulok szerkezete

Közös elemek

Mindhárom modul gyökérkönyvtárában megtalálható az alábbi három Python fájl.

A `config.py` nev fájlok a konfigurációs szkriptek, melyek célja, hogy futtatás eltt módosítani lehessen az elre beállított, nagybetsített globális változókban tárolt beál-

lításokat, mint például a feldolgozandó adat mennyiségét, az input fájlok adatait, vagy a prompt üzenetet.

A `main.py` a f 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 függséget behúznánk.

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

4.2.3. ModelInputPreparer

Ahhoz, hogy a modellek szemantikus képességeinek konzisztenciáját megvizsgálhassuk, elször 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 elemz modul létrehozása eltt szükség volt ennek a két funkcionalitásnak a létrehozására.

Az els modul a ModelInputPreparer (modell bemenet elkészít), 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 következ: A WiC adathalmaz minden sora 5, tabulátorral elválasztott értéket tartalmaz:

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

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

A modul hívási verme a következ:

```
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 elz modulban elállított kérdéseken.

Az elz modulban láthattuk, hogy a kérdések legyártása viszonylag egyszer fájl- és sztringfeldolgozó algoritmusokkal lehetséges. A modell futtatása viszont nem egy triviális feladat. Még kisebb méret Hugging Face modellek futtatása is rendkívül körülményes. Ez egy internetes API-kat és rendkívül sok idt és erforrást (CPU, GPU, RAM) igényl mvelet. Így nem meglep, hogy ennek a modulnak az elkészítése vette igénybe az id túlnyomó részét a szoftver elkészítésében.

A program elindítása eltt érdemes a `config.py` és a `modelname.py` beállításait ellenrizni és igény szerint módosítani. A modell tetszleges lehet, de a ajánlott a `modelname.py` fájlban található `supported\_models` listában lév modellek közül, annak is az els indexein elhelyezked modellek közül választani, mert ezek kompatibilisebbek a keretrendszerrel. A config fájlban lehetség van a promptnak az utasítás részén változtatni, amely aztán a kérdések elé kerül a prompt-felépítés során. Ebben lehet adni, hogy hány kérdésre válaszoljon, milyen kulcsszavakkal, terjedelemben, stb. A konzolra történ kiíratások mennyiségét is itt lehet állítani.

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

Ezután a program azt is leellenrzi, 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 szerepl 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 kezddik a 3. féltl 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 els szükséges csomag a torch, amellyel elkészítjük a hardvert a nagyméret adat számolására. A torch csomag no_grad() kontextuskezel 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. Els lépés a tokenizálás, amelyhez

segítséget nyújt a transformers AutoTokenizer osztálya, és annak from_pretrained metódusa, amely a promptunkat a modell számára érthet tokenekké bontja. Ezután következik a modell-specifikus rész, hiszen a különböz modelleknek eltér 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.

A modul hívási verme a következ:

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

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

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 els 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 elz modulokhoz hasonlóan választhat, hogy kilép a programból, vagy továbblép potenciálisan hibás input kérdésekkel. Ha a felhasználó megersíti, hogy tovább kíván lépni, a program megpróbálja helyreállítani az emiatt keletkez anomáliákat egy, vagy több sor törlésével.

A modul hívási verme a következ:

```
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 elz modulból inputként kapott modell válaszokon és soronként eldönti, hogy a "Yes", a "No", vagy a "?" (nem egyértelm) kategóriába sorolható-e be leginkább. Minden sor pontosan egy kategóriába soroltatik be. Az eldöntési algoritmusnak több módszere is van, ezért választható itt az `AnswerAwareClassificationRu`

és a `SentenceClassifier`, amelyet a config fájlban található `ADAPTIVE_RUN` változóval lehet beállítani. Ezután a kategóriákat a `TernaryResultsProcessor` összeveti a gold standardban szerepl 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 számszersíteni. Különösen igaz ez a `ModelOutputProcessor`, esetén, hiszen az eredmények könnyen számszersíthetk. Ez talán a legkönnyedebb módja a különböz futtatások összehasonlításának és ez teszi lehetvé a következtetések levonását. Éppen ezért minden eredményt egy 0 és 100% közé normalizált számként fejez ki a programom. Az eredményként az alábbi statisztikai indikátor mutatókat számítja ki a modul:

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

Konzisztencia

A program els lépésként végigmegy a válaszok sorain. Megnézi, hogy a modell mit válaszolt soronként. Ezt 3 kategóriába sorolja: `Yes`, `No` és `?`, azaz nem egyértelm, "talán". Ez utóbbi minden olyan válasz, amelynek nem egyértelmen "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-bl 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 mutatóban 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 egyszeren 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.

4.2.6. Bizonytalan

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

4.2.7. 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, konzisztens pontosság, kiegyensúlyozott pontosság és a tp,fp,fn,tn statisztikák.

4.3. Tesztelés

4.3.1. A programok futtatása

A PyCharm IDE kínál egyszer és felhasználóbarát futtatási módokat - a zöld háromszög ikonokat kell keresni, azzal lehet tetszleges .py kiterjesztés fájlt futtatni. Ez a projektem esetén is mködik. A keretrendszer main.py fájlokat kell elindítani a modulok lefuttatásához. Mivel azonban a projektem kódja sok fájlból, külss csomagokból és összetett projektszerkezetből áll, és a zöld gomb nem biztos, hogy megfelel 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 megfelel.

Futtatás 3 lépésben:
Elsként klónozzuk a projekt forráskódját a gépünkre, pl. a `~\PycharmProjects\` elérési útra. Ajánlott a projekt nevét "Thesis" néven hagyni. Következ 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. Egyszer 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" kiterjesztés fájlokban történik.

Hogyha több Python verziót is használunk, akkor annak is jelentsége 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.

4.4. Egyebek

4.4.1. Hibakezelés

A programban a hibák megtalálását és lekezelését egyszer try-catch blokkokkal [1] kezelem. Egyes hibák az eltér környezetekbl adódnak, pl. a Python interpreter eltér mappát tekint gyökérkönyvtárnak. Ezt is try-catch-csel oldottam meg, amellyel minden importot használó szkriptbe kétféle módon is megpróbálom beimportálni a függségeket.

4.4.2. Dupla importok

A moduljaimban szinte minden fájlban észrevehet, hogy az importok egy try-catch blokkban vannak megadva. Ez duplikált importálásnak tnhet, ám nem az, hiszen a verzérlési szerkezet biztosítja, hogy ha az els elérési út megtalálása meghiúsul, akkor a másik útvonalról töltdjenek 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.

4.4.3. 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, eltér architektúrájú és operációs rendszer gépen is teszteltem. Teszteltem Asus laptopon és különböz toronygépházas gépeken, Windows 10-en, 11-en és Linux Debianon is mködött a konzolos futtatás.

4.4.4. Skálázhatóság, bvíthetség

A szoftverem könnyen tovább bvíthet, hogy több és újabb modelleket is képes legyen támogatni, és terhelhet tetszlegesen nagy mennyiség bemenettel, anélkül, hogy lényegesebb futásid-növekedést tapasztalnánk, vagy lényegesebben 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 mködjön nagy méret, vagy érvénytelen input esetén is. Ezt elssorban a moduláris, és függvényekbe, osztályokba és metódusokba általánosított, absztrakt felépítésnek köszönheti a szoftver. Az egyes részek csak minimálisan függnek egymástól.

4.5. Jövbeli 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 elérhet lehessen. A Hugging Face Hub alkalmas platform lehet erre. Továbbá egy adatbázis a jelenlegi nyers szöveg-alapú adatkezelés helyett nagy elrelépés lenne.

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

5. fejezet

Kísérlet

5.1. A konzisztenciára vonatkozó megállapítások

Akkor beszélhetünk egyáltalán konzisztenciáról, tekinthetjük a konzisztens viselkedés els szintjét annak, ha a modell *reagál a bemenetre*, azaz a válaszának egyáltalán valami köze van a feltett kérdésekhez. Elfordul, hogy a modell teljesen vakon van és nincs reláció a válasza és az egyes kérdések között. Ekkor az összes mutatója 0.

A vizsgált nyelvi modellek jelents része nem képes stabil és konzisztens döntést hozni a WiC-feladat esetén. Ahelyett, hogy bináris osztályozást adnának, gyakran irreleváns generatív válaszokba esnek (pl. Hello, how can I help you today?), vagy repetitív tokenflodding jelenséget mutatnak, egy nagy valószínség 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 megfelelen az instrukciót, nem tartják a formátumot, és egyenes-fordított párok esetén extrém alacsony konzisztenciát adnak. Erre a megoldásom a `max_new_tokens` paraméter 1-re állítása volt a `transformers` könyvtár `_BaseModelWithGenerate` típusának a `generate` metódusában. Ez minden esetben garantálta, hogy a modell pontosan 1 tokenben, 1 szóban válaszoljon.

5.2. Méret- és konzisztencia-arányban megfelel nyelvi modellek kiválasztása és összehasonlítása

A modelleket a Témavezetm ajánlásával együtt választottam ki. ajánlotta, hogy a LMArena [**chiang2024chatbot**] és a Hugging Face [**wolf2019huggingface**] platformokon lév modelleket vizsgáljam meg, de a konkrét modell példány kiválasztását rám bízta. A modellek kiválasztásakor a cél az volt, hogy kellen kisméretek legyenek ahhoz, hogy gyorsan lefussanak a lokális környezetben, ám korlátozott paraméterszámuk ellenére a lehet legjobb "konzisztens pontosságot" tudják elérni, azaz minél több kérdésre konzisztensen és a gold standard szerint válaszoljanak.

Így a keretrendszeremet és a Colab jegyzetfüzetemet elssorban a

- Microsoft/Phi-4-mini-instruct [**phi4mini2024**],
- Google/Gemma-2-2b-it [**gemma22b2024**],

- Qwen1.5-1.8B-Chat [**qwen15chat2024**],
- Qwen/Qwen2.5-1.5B-Instruct [**qwen2.5**] és a
- Qwen/Qwen2.5-0.5B-Instruct [**qwen2.5**]

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. Az elnevezési szabály megtéveszt lehet, ugyanis gemma-*x*b... kezdet modellek els generációs változatok, míg a gemma-2-*x*b... a második, és a gemma-3-*x*b... a harmadik generációs modelleket jelöli.

5.3. Google Colab futtatás

A középs modul egyszersített változata elérhet Google Colab-on, a mellékletben található linken. 6

Mivel Google Colabot használtam, azon belül is GPU-s futtatókörnyezetet, így a kiértékelés megfelelen gyors volt. A webes környezet és a kikapcsolt valószínségi változók és az inferenciák egyesével történ 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 véletlentl,
- és izolált, azaz a modell egyik futtatásban sem "tud" semmit a korábbi bemeneteirl 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.

5.4. 2 konkrét modell kiértékelése 100 véletlenszer mintán a keretrendszerben

6. fejezet

Konklúzió

Nyilatkozat

Alulírott Fábián Bernát, programtervez informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem Informatikai Intézet Mesterséges Intelligencia Tanszékén készítettem, a programtervez 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 8.

. . . . . .
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 **témavezetmnek, Berend Gábornak**, hogy konzulensként és témavezetként 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 tekinthet meg.