Szegedi Tudományegyetem

Informatikai Intézet
Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszék

Nyelvi modellek teljesítményének összehasonlítása egy kontextusalapú szójelentés-felismerés (Word-in-Context) probléma megoldására és saját kiértékelő rendszer fejlesztése Pythonban

Comparing the Performance of Language Models on a Word-in-Context Benchmark and Developing a Custom Evaluation System in Python

Szakdolgozat

Készítette:
Fábián Bernát
Programtervező informatikus szakos hallgató

Témavezetõ:
Dr. Berend Gábor Iván
egyetemi docens

Szeged
2025

Feladatkiírás

A hallgató feladata a Word-in-Context probléma megismerése, a jelenlegi legjobb megoldások áttekintése, összehasonlítása, annak a megvizsgálása, hogy nyílt forráskódú generatív nyelvi modellek mennyire jól tudják megoldani a Word-in-Context feladatot determinisztikus és izolált válaszadás esetén, kóddal automatizálva a promptolást és a válaszfeldolgozást. Valamint egy angol nyelvű WiC kiértékelő rendszer megtervezése, implementálása és tesztelése.

- Ismerje meg az NLP és a nagy nyelvi modellek fogalomrendszerét.
- Általa választott modelleken vizsgálja meg, hogy mennyire érzékenyek arra, hogy

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

  formájú kérdésekben a 2 mondatot fordítva kapják meg, ahol w egy szó, $s1$ az első példamondat, és $s2$ a második példamondat. Tehát az "Ugyanazt jelenti-e a szó az A és B mondatban" eredménye ugyanaz, mint a "Ugyanazt jelenti-e a szó a B és A mondatban" eredménye.
- Tervezze meg, implementálja és tesztelje a kontextusalapú szójelentés-felismerő rendszerét, a WiC adathalmaz alapján. Az adatok letöltése után végezze el a szükséges adattranszformációkat, az adattisztítási feladatokat, szűrje ki a felesleges, vagy használhatatlan adatok körét, illetve készítse el az elemzésre a kész adatállományt.
- Értékelje ki a kontextusalapú szójelentés-felismerő rendszerén a teszthalmazt.
- Az elért eredményeiről készítsen statisztikai összesítéseket és látványos lekérdezéseket, ábrákat.

Elvégzett munkáit a Szakdolgozat keretében dokumentálja.

Tartalmi összefoglaló

- **A téma megnevezése**

  Egyrészt a kontextusalapú szójelentés-felismerés (Word-in-Context, WiC) feladat, tágabb értelemben a szójelentés-egyértelműsítés (Word Sense Disambiguation, WSD), nyelvtechnológiai (NLP) módszereket és eszközöket használva. Másrészt a generatív nyelvi modellek, abból is a nyílt forráskódú modellek teljesítménye, a promptolás és a válaszgenerálás automatizálása.

- **A megadott feladat megfogalmazása**

  A feladat megvizsgálni, hogy a Mohammad Taher Pilehvar által összeállított és publikált Word-in-Context feladatot önállóan választott ingyenes és nyílt forráskódú generatív nyelvi modellek mennyire jól oldják meg. A modellek válaszainak összehasonlítása és elemzése, ehhez egy saját adatfeldolgozó és kiértékelő rendszer megtervezése, implementálása Python nyelven, majd tesztelése. A rendszernek képesnek kell lennie a pontosság mellett a fedés és a precizitás értékét megadni, továbbá tévesztési mátrixot készíteni a predikciók eloszlásáról, az alábbi négy osztályba sorolva:

  - **TP (True Positive):** Azok az esetek, amikor a modell helyesen ismeri fel, hogy a szó jelentése megegyezik a két mondatban.
  - **FP (False Positive):** Azok az esetek, amikor a modell tévesen gondolja azt, hogy a szó jelentése megegyezik, pedig valójában eltér.
  - **FN (False Negative):** Azok az esetek, amikor a modell nem ismeri fel a jelentésazonosságot, pedig a szó jelentése valójában megegyezik.
  - **TN (True Negative):** Azok az esetek, amikor a modell helyesen ismeri fel, hogy a szó jelentése különböző a két mondatban.

- **A megoldási mód**

  - A szakdolgozatom első felében kielemeztem az eddigi legjobb modelleket és a legjobb módszereket szójelentés-értelmezés szempontjából és nyílt forráskódú nyelvi modelleket kerestem a LMArena és Hugging Face felületein.
  - Összehasonlító teljesítményvizsgálatot végeztem a Phi-4-mini-instruct, Gemma-2-2b-it és a Qwen1.5-1.8B-Chat között egy Google Colab szkriptben.
  - Saját adatfeldolgozó kiértékelő rendszert és függvénykönyvtárat készítettem a promptolás automatizálásához és a WiC probléma megoldására.

  - A választott modellek futtatása az automatizáló szkriptekkel különböző módokban a szósorrend-érzékenységének és egyéb metrikák összehasonlítására
  - A saját TF-IDF- és SenseBERT-alapú kiértékelés megtervezése, implementálása és tesztelése

- **Az alkalmazott eszközök, módszerek**
  - Google Colab és PyCharm, Python futtatókörnyezet
  - Git, GitHub
  - LMArena, Hugging Face
  - Google Colab
  - NLTK és WordNet
  - CUDA, Accelerate, Transformers, BERT
  - Google Táblázatok, Apps Script
  - AutoClicker.exe

- **Az elért eredmények** Sikerült átfogó elemzést nyújtanom 3 Hugging Face-ről letölthető generatív nyelvi modell teljesítményéről. Kiértékelő rendszer és függvénykönyvtár, amelynek célja a WiC (Word in Context) adathalmaz feldolgozása és a feladatok megoldásának vizsgálata különböző nyelvi modellek segítségével. Az alkalmazás arra is képes, hogy elemezze, mennyire érzékenyek ezek a modellek arra, ha a két példamondat sorrendjét felcseréljük.

  A másik eredményem egy, a mintafeladathoz készült feldolgozó algoritmuscsomag, amely elvégzi a szükséges adattranszformációkat, az adattisztítási feladatokat és az adat könnyen értelmezhető formába alakítását.

  Harmadik büszkeségem, hogy 60+ %-os eredményt értem el a módszeremmel, ami közelít a WiC: The Word in Context adathalmazra elért legjobb eredményekhez.

- **Kulcsszavak** WiC, WSD, NLP, LLM, nyelvi modell, szójelentés-ábrázolás

Tartalomjegyzék

Motiváció

Az informatika és a nyelvtechnológia fejlődésével a generatív mesterséges intelligencia mindennapjaink részévé vált. A nagy nyelvi modellek kiértékelésére számos benchmark, azaz teljesítményteszt, metrika fejlődött ki az évek során. A legnépszerűbb ilyen benchmark a SuperGLUE Benchmark, amely 8, komoly kihívást jelentő feladat elé állítja a nagy nyelvi modelleket. Ebben a papírban a WiC feladatot mutatom be.

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. A munkahelyeken – különösen IT területen – egy jól működő WiC modell jelentős előnyt biztosíthat kódok, dokumentumok, ügyfélkommunikáció vagy akár jogi szövegek értelmezésében, ami kulcsfontosságú lehet például a tudásközpontokban és startup cégeknél is. Egy hatékony WiC modell tehát nemcsak tudományos értékkel bír, hanem gyakorlati alkalmazásokban is közvetlen hasznot hozhat, főleg a nyelvészeti informatikai területen dolgozók számára.

Bevezetés

Irodalmi áttekintés

A szóbeágyazások tervezésükből adódóan képtelenek modellezni a szavak szemantikájának dinamikus természetét, vagyis azt a tulajdonságot, hogy a szavak potenciálisan különböző jelentéseknek felelhetnek meg. E korlátozás kezelésére tucatnyi specializált jelentésreprezentációs technikát javasoltak, mint például a jelentés- vagy kontextualizált beágyazásokat. Azonban a téma kutatásának népszerűsége ellenére igen kevés olyan értékelési viszonyítási alap létezik, amely kifejezetten a szavak dinamikus szemantikájára összpontosít. A meglévő modellek túllépték az erre a célra szolgáló standard értékelési adatkészlet, azaz a Stanford Kontextuális Szóhasonlóság teljesítménykorlátját. A megfelelő viszonyítási alap hiányának kezelésére készült a nagyszabású, szakértők által összeállított annotációkon alapuló Word in Context (Szó a Kontextusban) adatkészlet, a WiC. A WiC nyilvánosan elérhető a WiC: The Word-in-Context Dataset weboldalon.[1]

A szakdolgozat célja

A szakdolgozatom elsődleges célja, hogy megvizsgáljam egy pár gondosan megválasztott modellen, hogy mennyire jól oldják meg a WiC feladatot, és mennyire érzékenyek arra, ha a mondatokat felcserélve kapják meg. Tehát mennyire befolyásolja a döntésüket a szavak sorrendje. Mindezt úgy, hogy az egyetlen kóddal futtatható és jól dokumentálható legyen, és a kérdéseket egyesével kell beadni, hogy az előző válaszok ne befolyásolják a modell döntését, csak a saját tudására, tehát az előtanítása alatt szerzett információra támaszkodjon.

A dolgozat célja még a kontextuális szóbeágyazások és a generatív nyelvi modellek mély megismerése, majd egy WiC kiértékelő rendszer létrehozása, amely különböző módszereket hasonlít össze, beleértve a hagyományos TF-IDF módszereket, Lesk algoritmust és WSD technikákat, valamint modern neurális megközelítéseket. A szakdolgozatom első felében azt vizsgálom meg, hogy népszerű nagy nyelvi modellek mennyire jól oldják meg a WiC problémát.

Végül saját modellt hozok létre, amely képes legalább 60%-os pontosságot elérni a teszt adathalmazon, emellett a fedést, a precizitást és az F1 pontszámot megadni, tévesztési mátrixot készíteni a TP, FP, FN, TN címkék eloszlásáról.

1. fejezet

A WiC probléma

1.1. 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. A többértelműséget idegen szóval ambiguitásnak nevezik.

1.2. A WiC probléma

Az emberi nyelvekben a többértelmű szavak attól függően, hogy milyen szövegkörnyezetben fordulnak elő, több, esetenként egymással össze nem függő dolgot is jelenthetnek. 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 képességei fejlesztésére.

1.3. A WiC feladat és jelentősége

A WiC feladat lényege, hogy egy adott szó két különböző mondatbeli előfordulásáról eldöntse, hogy azonos értelemben szerepel-e. A természetes nyelvi feldolgozásban [1] ez kulcsfontosságú, mivel segít a szövegértelmezésben és a jelentésalapú keresés fejlesztésében.

1.4. Egy rendszer feladata a WiC adathalmazon

Egy rendszer feladata a WiC adathalmazon 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.

1.5. Bináris osztályozás

A WiC feladat egy bináris osztályozási (binary classification) problémaként van megfogalmazva: el kell dönteni, hogy egy adott szó két különböző mondatbeli előfordulása ugyanabban az értelemben szerepel-e. A bináris osztályozási feladatok problémakörében is változnak a trendek olyan szempontból, hogy egyre inkább a neurális hálók és nagy nyelvi modellek az elterjedtek bináris osztályozási feladatokra is. A Lesk-algoritmus [2] egy klasszikus szóértelem-felismerő módszer, amely a szótári definíciók és a kontextus összevetésével próbálja meghatározni a szó legmegfelelőbb jelentését. A legjobbak mégis az olyan, kifejezetten emberi szöveg megértésére specializálódott nagy nyelvi modellek, mint például a GPT-4.5 és a Gemini-2.5 - 2025 tavaszán.

1.6. A WiC adathalmaz

A Word in Context adathalmaz egy jó minőségű, nyelvészeti szakértők által készített benchmark adathalmaz. A mondatok a WordNetből, a VerbNetből és a Wikiszótárból származnak. A WiC adathalmaz segítségével megvizsgálhatjuk, hogy egy rendszer

[1] Természetesnyelv-feldolgozás, számítógépes nyelvészet és nyelvtechnológia néven is hivatkoznak rá, angolul NLP

(modell, algoritmus) mennyire képes a szavak jelentését megérteni különböző kontextusokban. A WiC feladat része a SuperGlue [3] Benchmarknak, amely egy széles körben elfogadott benchmark nyelvi modellek kiértékelésére. 8 feladatból áll:

- BoolQ (Boolean Questions)
- CB (CommitmentBank)
- COPA(Choice of Plausible Alternatives)
- MultiRC (Multi-Sentence Reading Comprehension)
- ReCoRD(Reading Comprehension with Commonsense Reasoning Dataset)
- RTE(Recognizing Textual Entailment)
- **WiC(Word-in-Context)**
- WSC (Winograd Schema Challenge)

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

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

1.1. táblázat. A WiC adathalmaz tesztkészletének néhány bejegyzése

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.

1.6.1. Eddigi legjobb megoldások a WiC-re

A feladatra bőven érkeztek megoldások, számos módszerrel értek el 60% feletti eredményt. 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.

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. Forrás: The Word-in-Context Dataset

1.7. Nagy nyelvi modellek

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

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

Forrás: Wikipédia

1.7.1. Chatbot Arena, vagy LMSYS - nagy nyelvi modelleket összehasonlító platform

Kutatómunkám első lépéseként utánanéztem, hogy melyik modellek a legjobbak angol nyelvű instruction following feladatokban. Ehhez a LMArenát [5] megfelelő referenciának tartottam, mert ez az oldal igen lenyűgöző módon minden ismertebb ingyenes nagy nyelvi modellhez API-n keresztüli limitált hozzáférést biztosít: 2025. május 16-án 101 modell API-ja volt kipróbálható az oldalon keresztül több tucatnyi cégtől, a ranglétrán pedig 241 modell teljesítményét mérhetjük össze különböző kategóriák alapján emberi szavazatok alapján, amely több mint 1.5 millió ember szavazatán alapszik. A folyamatosan frissülő rangsor menüben az opciók közül az "Instruction Following" és az "English" kategóriát vizsgáltam meg, mert a problémám ehhez a területhez áll a legközelebb.

Az oldalon található modellek mind elérhetőek és kipróbálhatóak, és általában saját, felhasználóbarát webes felületük is van, amely személyre szabható, elmenthetőek a korábbi beszélgetéseink, és egyéb kényelmes szolgáltatásokkal kecsegtetnek. Azonban fontos észben tartanunk, hogy ezeket a felületeket fejlesztő cégek bármilyen publikus webes forrást felhasználhatnak a modelljeik tanításához, beleértve a chatben megadott információt. A nagy cégeknek, mint például a Google-nek, az OpenAI-nak és a DeepSeek-nek is vannak nyílt forráskódú, jó minőségű modelljeik, melyek hatalmas adathalmazokon be vannak tanítva. Például a GPT-3 tanítása során a Common Crawl Wikipédiát, WebText2 reddit posztokból és linkjeiből kinyert adatokat, internetes könyveket és az angol nyelvű wikipédiát is felhasználták. [6] Régebben a New York Times termékeit is nagy mennyiségben használták, ám a hírlapíró cég ezt felhasználási feltételeiben megtiltotta. [7]

A nagy tech cégek, mint a Meta és az X, felhasználják a közösségi médiára feltöltött adatainkat a modelljeik fejlesztéséhez. A Google pedig a felhasználási feltételeit is többször megváltoztatta, hogy felhasználhassa szolgáltatásainak felhasználói adatait az AI termékei fejlesztéséhez. [8]

A Hugging Face [9] [több mint 1.7 millió modellt] és [csaknem 400 ezer adathalmazt] kínál különféle gépi tanulási feladatokra, beleértve a természetes nyelvfeldolgozást, képfeldolgozást és kódgenerálást is. Ezek a modellek nyilvánosan elérhetők és szabadon használhatók. Előnyük, hogy nyílt forráskódúak, így tetszőlegesen módosíthatóak - habár alapos szakértelem kell hozzájuk. Emellett biztonságosak: letölthetőek saját gépre, kódból futtathatóak anélkül, hogy egy szerverrel kommunikálnának. Emiatt privát adatot is be lehet nekik adni. Jellemzően minél kisebb a paraméterszámuk, annál gyengébb a visszaemlékező képességük, rövidebb válaszokat részesítenek előnyben, és annál gyakrabban követnek el hibákat vagy nyelvi torzulásokat - például nyelvtani hibákat, kódban szintaktikai hibákat, inkoherens (összefüggéstelen, logikailag következetlen) válaszokat. A kvantálás általában csak a népszerűbb/nagyobb modelleknél (pl. Llama, Gemma, Mistral) elérhető, és gyakran közösségi kontribúciók eredménye (nem hivatalos verziók). A kvantálás a modell súlyait alacsonyabb bitmélységűre (pl. 32 $\rightarrow$ 8/4 bit) konvertálja, ami csökkenti a memóriaigényt és gyorsítja a következtetést. Emiatt könnyen félrevezethetőek és könnyen hallucinálnak is. Összefoglalva hátrányaik a

- korlátozott kontextusmegértés a csökkentett bitmélység miatt, ezért nehezebben követik a hosszú gondolatmeneteket,
- gyenge memóriájuk, elfelejtik a korábbi információkat,
- és túlzott általánosítás is jellemző rájuk.

Fogalmak

Ahhoz, hogy nagy nyelvi modelleket ki tudjunk értékelni, meg kell ismernünk néhány alapfogalmat.

1.7.2. Mi a prompt?

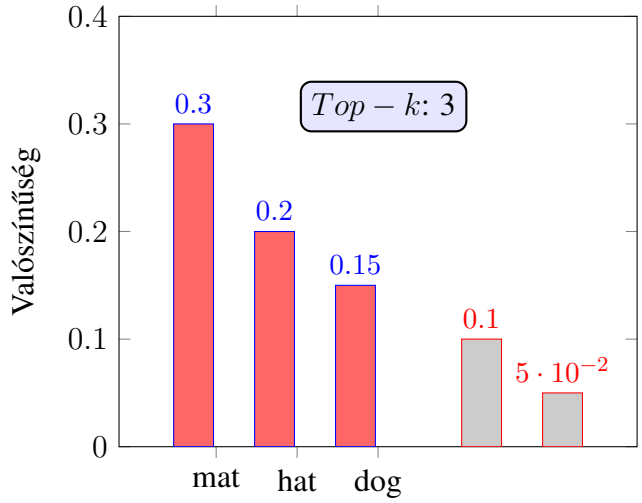
A prompt az informatikában egy nagy nyelvi modell számára beadott input, amelyre a modell egy előre meghatározott kimenetet generál. Működési elve lényegében ugyanaz, mint egy hagyományos konzolnak, parancssori interfésznek, például egy Windows rendszeren a PowerShell, vagy Linuxon Bash terminállal, azonban prompt esetén a kommunikáció emberi nyelven történik. 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.

1.7.3. Inferencia

A modell promptra adott válaszát gyakran inferenciának nevezik, ami következtetést jelent. Ez arra utal, hogy a modell a megtanult minták alapján logikailag következtet a legvalószínűbb kimenetre, nem pedig szó szerint "visszadobja" a betanított adatokat.

1.7.4. Determinisztikus következtetés, temperature, top-p és top-k

A kísérletemnek csak akkor van értelme, ha a futtatás determinisztikus. Azt jelenti, hogy a modell mindig ugyanazt a kimenetet adja ugyanazon bemenet esetén, mivel a véletlenszerűséget kizáró paraméterekkel (pl. $temperature = 1.0, top{-}k = 0, top{-}p = 1.0$) működik. A temperature (hőmérséklet) 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) [10].



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.

1.7.5. 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** (angolul *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-3 alapértelmezett kontextusablaka 2048 token, míg a GPT-4 esetében ez elérheti a 8192 vagy akár 32 768 tokent is, verziótól függően.

2. fejezet

Legkonzisztensebb nagy nyelvi modellek megtalálása

Konzisztencia

A kutatásom első felének célja a modellek teljesítményének összehasonlítása a WiC-ből szedett kérdéseken volt, de nem az érdekelt elsősorban, hogy minél több kérdésre a gold standard [1] szerint válaszoljanak, hanem hogy a szavak sorrendje ne befolyásolja a válaszadásukat. Tehát 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 válaszolják (változatlan w, $s1$ és $s2$ esetén, ahol w egy szó, $s1$ az első példamondat, és $s2$ a második példamondat).

Először a LMArena és általam ismert legjobb modelleket vizsgáltam meg, hogy mennyire érzékenyek arra, hogyha a 2 mondatot felcseréljük. A WiC adathalmazból előállított kérdésekből szúrópróbaszerűen adtam be kérdéseket a vizsgált modellnek, majd töröltem a modell memóriáját, és ezt a kérdést a 2 mondatot felcserélve is elküldtem a modellnek. A mondatokat a Peternity 4 Python programommal generáltam, erről majd később írok. Végül egy Google Táblázatban a válaszokból összesítettem az egyes promptolási módszerek pontosságát, kiegyensúlyozott pontosságát, hogy mekkora arányban egyeznek a válaszpárok. A teszthalmazban már alapból randomizált sorrendben vannak a kérdések, így az első 60 beadását megfelelőnek találtam.

$$\text{pontosság} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

$$\text{kiegyensúlyozott pontosság} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{2.2}$$

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

Hugging Face modellek

Olyan modellt szerettem volna választani az összehasonlításhoz, amelyek nyílt forráskódúak, nagyjából azonos, másfél milliárd és 5 milliárd közötti paraméterszámúak, de legfeljebb 2 milliárd paraméterszám-különbség van köztük. A Google Colab webes Python futtatókörnyezetet, Hugging Face modelleket, és a nagy cégek modelljeinek webes API-jait használtam.

Először is, ahhoz, hogy egyértelmű eredményt kapjak a modell teljesítésére, ahol csak lehetett, minden véletlenszerűséget biztosító paramétert ki kellett kapcsolnom. Erre azért volt szükség, hogy ne a szerencsén múljon az, hogy a modell két válasza megegyezik, vagy eltér, hanem a modell felépítésén. A sztochasztikus faktorokkal ugyanis ugyanarra a kérdésre eltérő futtatáskor eltérő választ adhat egy modell, ami nem vezet hiteles kutatáshoz. A LMArena és a Google AI Studio oldalán van erre lehetőség.

Hogy ezt elérjem, a *temperature* értékét 0-ra, a *top-p* értékét pedig 1-re állítottam. Ezáltal a válaszok sztochasztikus változatosságát minimálisra redukáltam.

Egy példa kérdés:

Egyenes sorrendben

```
Does the word "defeat" mean the same thing in sentences
"It was a narrow defeat." and "The army's only defeat."?
```

Fordított sorrendben

```
Does the word "defeat" mean the same thing in sentences
"The army's only defeat." and "It was a narrow defeat."?
```

A véletlen faktor kikapcsolása mellett az is fontos volt, hogy egyesével adjam be a kérdéseket. Amikor ugyanis egyszerre N kérdéssel bombáztam a modelleket, akkor az is egy figyelembe veendő tényező volt, hogy az N kérdés milyen sorrendben lett feltéve. Ráadásul a modellek, ha egyszerre túl sok kérdést kaptak, jellemzően csak 40-60 kérdésre válaszoltak, az utána lévőket teljesen ignorálták. Ezzel szemben, ha minden kérdés esetén tiszta lappal indítottam, és új, független promptként adtam be, akkor ez a hatás nem tudott bezavarni. Emiatt tisztább a kísérlet, ha a modell nem egyszerre kapja meg mind az N mondatot, amiről a döntést várjuk, hanem N darab olyan kérdésként, amelyekben egyenként csak pontosan egy kérdés szerepel.

Ezt a fajta módszert, hogy a webes felületek input mezőjébe adtam be a kérdéseket, csak API-alapú automatizációval lehetne automatizálni, de a webes felületek nem erre készültek. Ahhoz, hogy automatizáljam a promptolást, egy, a témavezetőm által készített és javasolt Google Colab jegyzetfüzetet [11] fejlesztettem tovább. Ez egy egyszerű, de hatékony jegyzetfüzet, amellyel Hugging Face token birtokában tetszőleges modell válaszadásra bírható, és mindez hatékonyan automatizálható. Hogy ezt elérjem, 3 kérdéssort készítettem.

- Egyrészt egy könnyű feladatot készítettem: egy saját kérdéshalmazt, amely egy 19 nagyon egyszerű kérdésből áll.

- Másrészt a WiC adathalmazra alapuló kérdéshalmazomból, főleg a teszt részosztályból szúrópróbaszerűen randomizált kérdéseket tettem fel és adtam be a modelleknek, 60 darabot.

- Harmadrészt, szintén a kérdéshalmazomból, főleg a teszt részosztályból kiválasztottam pár hasonló szót és párat a legegzotikusabb szavak közül, vagy szövegkörnyezetet tartalmazó mondatot, amelyekkel könnyű a modellt összezavarni.

2.1. A megfelelő nyelvi modellek kiválasztása

A nagy AI cégek, mint az OpenAI, Anthropic és a DeepSeek olyan fejlett általános célú generatív mesterséges intelligencia modelljeik vannak, amelyek minden kategóriában csúcsteljesítményt nyújtanak, vállvetve a Google, Microsoft, X és Meta technológiai óriásokkal, csak hogy párat említsek. Azonban a feladatom megfogalmazásában szerepelt, hogy ingyenes és nyílt forráskódú modelleket kell megvizsgálnom, így ezek többsége nem jöhetett szóba. Kíváncsiságból azonban mégis megvizsgáltam a következő modelleket:

- ChatGPT (OpenAI) – Gyors, sokoldalú, fizetős API

- Claude (Anthropic) – Biztonságközpontú, fizetős API

- DeepSeek (DeepSeek) – Hatékony, tömör, nyílt forráskódú, nagyméretű modellek

- Gemini (Google) – Innovatív, integrált

- Grok (X) – Valós idejű adatfeldolgozó

Modell neve	**Helyes válaszok (db/összes)**	**Kiegy. pontosság** (%)	**Konzisztencia (%)**
GPT	143/188	76	90
Claude	138/188	73	87
DeepSeek	141/188	75	83
Gemini	156/188	82	91
Grok	126/188	67	84

2.1. táblázat. Nagy nyelvi modellek teljesítményértékelése. A 188 kérdés a 19 előre kiválasztott könnyű, 60 random, valamint 15 előre kiválasztott nehéz kérdésből álló korpusz egyenes és fordított beadásából tevődik össze.

Hogy a promptolást automatizáljam, Hugging Face-en elérhető modelleket futtattam a Google Colab szkriptemben 4.5.
A témavezetőm által ajánlott modellek közül a

- Microsoft/Phi-4-mini-instruct [12],

- Google/Gemma-2-2b-it [13] és a

- Qwen1.5-1.8B-Chat [14]

modelleket választottam, mert ezek elég jók méret/minőség arányban a közösségi média fórumok szerint. Közelítő arányban sincsenek a nagy AI és tech cégek top modelljeivel, de a méretükhöz képest gazdag szókinccsel és jó nyelvérzékkel rendelkeznek. Mind a hármat könnyű, közepes és nehéz kérdéseken, mind a 3 kérdésnehézséget egyenes és fordított sorrendben teszteltem. Ez így összesen `3*3*2=18` kiértékelést jelentett. Hangsúlyozom, hogy nem a gemma-2b-it, hanem a gemma-*2*-2b-it változatot választottam. Az elnevezési szabály kissé összezavaró, ugyanis gemma-*x*b... kezdetű modellek első generációs változatok, míg a gemma-2-*x*b... a második, és a gemma-3-*x*b... a harmadik generációs modelleket jelöli. A gemma 3 család 2025 tavaszán tétetett közzé, így erről átfogó vizsgálatot nem végeztem, ám feltehetően jobb a szójelentés-felismerés feladatokban, mint elődje. A modellek eredményeit összessítő táblázat a 4.1. táblázatban látható.4.5

A nagy nyelvi modelleket futtató szkriptem nyilvánosan megtekinthető, és a kísérlet elvégezhető Google Colab-on, a mellékletben található linken. 4.5

Mivel Google Colabot használtam, azon belül is GPU-s futtatókörnyezete, így a kiértékelés megfelelően gyors volt. A webes környezet és a kikapcsolt valószínűségi változók miatt elmondható, hogy a kiértékelésem módja:

- reprodukálható, azaz bárki meg tudja ismételni a kísérletet
- determinisztikus, tehát ugyanarra az inputra mindig ugyanaz lesz az output, nem függ a véletlentől.

2.1.1. Google AI Studio

A Google AI Studio egy ingyenes, Google által fejlesztett platform, ahol megtalálható az összes 'state-of-the-art' modelljük, és rendkívül személyre szabhatóak. A modellek kipróbálgatása után arra a következtetésre jutottam, hogy a Gemini 2.5 Pro Experimental 03-25 a legjobb a WiC feladatra. Egy inferenciában, kevesebb mint 10 perc alatt képes válaszolni mind az 1400 tesztkérdésre, amelyekre több futtatásom esetén is közel 80%-os pontossággal teljesített, mind egyenes, mind fordított sorrendben beadva a kérdéseket, átlagosan 91%-os egyezéssel az egyenes és a fordított sorrendben adott válaszai alapján. Lehetségesnek tartom, hogy a sikerességéhez hozzájárul az, hogy látta a kérdéseket az előtanulítása közben, és hatékonyan vissza tudott rá emlékezni, de az mindenképpen figyelemreméltó, hogy a kézi emberi kiértékeléssel egyező hatékonyságot egyedül ez a variáns ért el.

Összességében a legkonzisztensebbek a tesztjeim alapján a Gemini-2.5 variánsai, a GPT-4 és a GPT-4.5o, ám ez csak empirikus vélemény.

3. fejezet

Három generatív nyelvi modell összehasonlítása

A gemma-2-2b-it, Phi-4-mini-instruct és a Qwen1.5-1.8B-Chat összehasonlítása

3.1. Módszer

3.1.1. Előkészítés

A generatív nyelvi modelleket egy Google Colab szkriptben hasonlítottam össze. Az projektem többi részét PyCharmban készítettem, ám itt a Colabra volt szükségem, mert lokálisan nagyon nehéz futtatni egy több billió paraméteres nyelvi modellt. Annál kisebbet pedig nem tartottam megfelelőnek, mert nagyon rosszul teljesítenének a WiC feladatra. Az első félévben lokálisan próbáltam futtatni "kisméretű" modelleket, azonban GPU-, RAM- és tárhelyproblémákkal küszködtem. Szerencsére a Google ingyenesen biztosít lehetőséget Cloud-alapú futtatásra, és GPU-t is biztosít. A modell egy távoli gépre töltődik le, és GPU-t is használ megfelelő beállításokkal.

3.1.2. A beadott szöveg formátuma

A modelleket úgy teszteltem, hogy eldöntendő kérdéseket tettem fel nekik egyesével:

Angolul	**Magyarul**
Does the word *w* mean the same thing in sentences *s1* and *s2*?	Ugyanazt jelenti-e a *w* szó az *s1* és *s2* mondatban?

Ahol *w* egy szó, *s1* az első példamondat, és *s2* a második példamondat.

3.1.3. A modelleket tesztelő kérdések

Alapos tesztelés után azt találtam, hogy nem célszerű a teljes nyers adathalmazból előállított kérdéseket beadni a modelleknek, helyette 3 gondosan összeállított kérdéshalmazzal értékeltem ki a teljesítményüket.

- Egyrészt egy könnyű feladatot készítettem: egy saját kérdéshalmazt, amely egy pár nagyon egyszerű kérdésből áll. Ha egy modell ezeket sem tudta jól megoldani, akkor a másik kettőt nem értékeltem ki rajta.
- Másrészt a WiC adathalmazra alapuló kérdéshalmazomból szúrópróbaszerűen randomizált kérdéseket tettem fel és adtam be a modelleknek, főleg a teszthalmazból.
- Harmadrészt, szintén a kérdéshalmazomból kiválasztottam pár hasonló szót, vagy szövegkörnyezetet tartalmazó mondatot, amelyekkel könnyű a modellt összezavarni.

A WiC megfelelő referenciaadatkészlet erre is, hiszen szakértők által gondosan annotált példákon alapul. Könnyen lehetséges, hogy már látták a tesztelt modellek (Phi, Gemma, Qwen) a WiC adathalmaz részét vagy egészét, ám nem tudnak rá visszaemlékezni, mert több milliárd mondatot láthattak a tanításuk során.

A WiC adathalmazban található mondatokban a szavak, mondatjelek és írásjelek tokenizálva vannak, szóközzel vannak elválasztva a megelőző és következő szótól, mondatjeltől és írásjeltől. Így a vessző, mondatvégi pont, az angol birtokos személyjel 's rag és bármilyen aposztrófot használó angol rag, jel, képző könnyen kiemelhető a szövegből. Emiatt viszont külön figyelnem kellett, amikor természetes nyelvezetűvé alakítottam a szöveget, hogy ezeket is gondosan átalakítsam.
Erre használtam a `sentence_normalizer` szkriptem függvényeit 4.5.

Emellett volt pár kifejezés és speciális karakter, amelyet ahhoz, hogy Python szótárként eltároljam, escape-elnem kellett, vagyis el kellett érnem, hogy a Python interpreter egyszerű karakterként kezelje őket, és ne program utasításként. A mondatokat ugyanis string dictionary-ként tároltam el, amelyekben az idézőjeleknek és kettőspont (:) karaktereknek speciális funkciója van. Így az alábbi helyzetekben gondosan átalakítottam a szöveget a Python szintaxisnak megfelelő alakba. Ezt is a `sentence_normalizer` szkriptem függvényei végezték.

Eredeti	Átalakított
My boss is out on another of his three martini lunches. Will you join us at six o'clock for martinis?	My boss is out on another of his three martini lunches. Will you join us at six o\'clock for martinis?
The derivative of f : f(x) = x^2 is f':f'(x) = 2x. 'electricity' is a derivative of 'electric'.	The derivative of f f(x) = x^2 is f\' f\'(x) = 2x. 'electricity' is a derivative of 'electric\'.

3.1. táblázat. A `sentence_normalizer` függvény átalakítása

Én ugyanis a kutatómunkám első fázisaként a weben, távoli szerveren futó modelleken szerettem volna tesztelni, ahova a mondatok ilyen formában nem feleltek meg a kiértékelésre, mert pont az volt a célom, hogy megvizsgáljam, mennyire értik meg jól a nagy nyelvi modellek az emberi szöveget. Emiatt a listában felsorolt tokeneknek az előfordulásainál átalakítottam a szöveget: kitöröltem az előttük lévő szóközt. Ezzel lényegében

visszaalakítottam a mondatokat emberi nyelvre. Ezt reguláris kifejezéseket használva és mintákat keresve Python algoritmussal oldottam meg.4.5

3.1.4. Kérdés-válasz szótár

A naiv kérdés-válasz készítési ötlet az lehetne, hogy a vizsgált szó legyen a kulcs, és a kérdés a válasz. Ez azonban nem célszerű, sőt hibás, ugyanis egy szóhoz több bejegyzés is tartozhat az adathalmazban, viszont a mondatpárok egyediek egymáshoz képest. Emiatt a kérdéseknek kell lenniük a kulcsoknak, amelyeknek a gold standard válasz az értéke. Például egy szótárelem kulcs-érték pár a következőképpen néz ki:

```
Key: Does the word "{word}" mean the same thing in
"{sentence1}" and "{sentence2}"?
Value: Yes VAGY No
```

3.1.5. Futtatás

A szkriptem leírása:
A modell nevének a pontos Hugging Face azonosítóját kellett megadni, a

```
model_name = <company/model-name>
```

utasítással adtam meg. Az

```
AutoTokenizer.from_pretrained(model_name)
```

sorral a tokenizer példányt hozom létre. Ez az eszköz felelős azért, hogy a nyers szöveget a modell által értelmezhető tokenekre bontsa. A `from_pretrained` hívással automatikusan letöltöm a modellhez illő tokenizálót. Az

```
AutoModelForCausalLM.from_pretrained(...)
```

függvény tölti be a nyelvi modellt. Amely képes szöveget generálni.

- `device_map='auto'`: Automatikusan eldönti, melyik modelltömb kerüljön melyik eszközre (pl. GPU, CPU).
- `torch_dtype='auto'`: Automatikusan kiválasztja a megfelelő adattípust a modellhez (pl. float16, bfloat16).
- `trust_remote_code=True`: Ezzel engedélyezem a modellhez tartozó egyedi kód futtatását, ami akkor kell, ha a modell nem a standard Hugging Face interfészt használja.
- `offload_buffers=True`: Ez különösen akkor hasznos, ha a GPU memóriája nem elég a teljes modell betöltésére. Ilyenkor a kevésbé használt rétegek puffereit

a rendszer a CPU-ra offload-olja, vagyis átrakja, hogy kevesebb GPU-memóriát használjon. Ezt egy figyelmeztetés javasolta, amit így beépítettem.

- `do_sample=False`: Teljesen determinisztikussá teszi a futtatást. Kikapcsolja a sztochasztikus mintavételezést, és átvált greedy decoding üzemmódba (amikor a modell mindig a legnagyobb valószínűségű következő tokent választja ki).

A tokenizálást az `apply_chat_template` végzi el, amely egyesíti a különböző modellek output formátumát. A `Gemma` esetén a `flash-attention` csomag telepítése szükséges. Paraméterei:

```
return_tensors='pt'         # A kimenetet PyTorch tensor
   formatumban kerem.
add_generation_prompt=True # Specialis jelzes, hogy generalni
   kell valaszt.
```

Ezután a `.to(model.device)` biztosítja, hogy az input tensor ugyanarra az eszközre (pl. GPU) kerüljön, mint a modell.

A szöveg-generálás lépése:

```
output = model.generate(
    inputs,
    max_new_tokens=200,      # Maximum 200 uj token
    do_sample=False,        # Determinisztikus valasz
)
```

A válasz dekódolása:

```
decoded = tokenizer.batch_decode(output, skip_special_tokens=
   True)

for reply in decoded:
    print(reply)
```

`skip_special_tokens=True` eltávolítja az olyan vezérlő tokeneket, mint `<s>` vagy `<|endoftext|>`, amelyek nem részei a természetes szövegnek.

Végül egy for ciklus segítségével az összes dekódolt választ szépen sorban kiírom:

```
    for i, sentence in enumerate(decoded):
    print(f"[{i}] {sentence}")
```

A blokkot `torch.no_grad()` kontextusba helyeztem. Enélkül nagyon pazarolná a program a GPU memóriát, és rövid úton "out of memory" hibába futna. A szkripttel hibátlanul le tudtam tesztelni a modelleket, amelyeket szerettem volna.

3.1.6. Kiértékelés

A nyelvi modellek hajlamosak az önismétlés hibájába esni, és az is gyakori torzulás, hogy az egyik osztályt aránytalanul preferálják a másik fölött, tehát például a WiC kérdéseinek 80%-ára nemmel válaszolnak. A kiértékelés során figyelembe vettem, hogy ha mindegyik kérdésre ugyanazt a választ adja, és ezzel aránytalanul sokat eltalál, az ne érjen annyi pontot, mint ha közel fele-fele arányban tippel vegyesen. Más szóval, ha egy modell mindenre ugyanazt válaszolja, és történetesen ezzel jól lefedi az egyoldalú gold standardot, az nem jelent valódi tudást, csak szerencsés torzítást. A kiegyensúlyozott pontosság (balanced accuracy) módszerével az abszolút `true positive` arány helyett egy átlagolt `true positive` arányt adunk meg az egyes osztályokra.

Ez segít, ha pl. a gold standard 80% "Yes" és 20% "No", de a modell mindig "Yes"-t mond – a sima pontosság 80%, de a kiegyensúlyozott pontosség csak 50%.

3.1.7. A modell válaszainak bővítése

Hamar be kellett látnom, hogy az algoritmus egy puszta "Yes" vagy "No" válasza önmagában nem sok információt rejt magában. Ha csak ennyit kapok vissza, nem tudom, hogy hogyan gondolkodott a modell, milyen szóvektorai vannak. Emiatt, hogy a puszta "Yes" és "No" válasznál többet kapjak vissza, többféle módszert alkalmaztam:

1. Megkértem a modellt, hogy írja le a gondolatmenetét.
2. A promptban utasítottam a modellt, hogy érveljen is, miért tartja helyesnek a válaszát.
3. Megkértem, a modellt, hogy egy *normalizált skálán*, mondjuk egy 0 és 100 közötti számmal, a határértékeket beleértve ítélje meg, hogy a modell mennyire biztos abban, hogy a célszó a 2 mondatban ugyanabban az értelemben van jelen. 100% jelenti azt, hogy a modell teljesen biztos abban, hogy a célszó a 2 mondatban ugyanazt jelenti, 0% esetén pedig teljesen biztos az ellenkezőjében.

```
Answer with "Yes" or "No" with reasoning and your confidence
    score of "Yes" in percentage. 100% means you are a
   hundred percent sure that they mean the same thing, 0%
   means the opposite.

Does the word "{word}" mean the same thing in sentences "{
   sentence1}" and "{sentence2}"?
```

```
17      print(f'Answer all {len(selected_questions)} questions with Yes or No{with_reasoning}!')
```

3.1. ábra. A nagy nyelvi modellek promptját úgy kezdtem, hogy érveljen minden mondat esetében.

Tapasztalatom szerint ez elég volt ahhoz, hogy részletes válaszokat kapjak a modellek webes futtatása esetében, de lokális futtatásnál is segített tájékozódni.

Egyszavas válaszok

A szótárból előállított prompt formátuma pedig a következő:

```
Answer all `n` questions with Yes or No!
Does the word "defeat" mean the same thing in sentences
"It was a narrow defeat." and "The army's only defeat."?
Does the word "groom" mean the same thing in sentences
"Groom the dogs." and "Sheila groomed the horse."?
.
.
.
(n sor összesen)
```

A modellek erre a formátumra szinte mindig csak egy Yes vagy No szóval válaszolnak, ezt saját teszteléssel is megerősítettem. Ezeket az előrejelzéseket egy egyszerű `Yes -> T, No -> F`leképezéssel össze lehet hasonlítani a gold fájlokban található igazat és hamisat jelölő T és F címkékkel.

Eltérő kimenetet kapok, ha érvelés kérésével adom be a modelleknek a szöveget:

```
Answer all `n` questions with Yes or No with reasoning!
Does the word "defeat" mean the same thing in sentences
"It was...
```

Ebben az esetben a chatbotok 1-2 mondatban meg szokták magyarázni a döntésük mögötti logikai érvelést. Ez is hasznos információ, amelyet felhasználok a modell teljesítményének megítéléséhez.

A modellek válaszát egy Google táblázatba mentettem el. Az első oszlopban jegyeztem fel a szavakat az adathalmazokból, a másodikban a gold standardot, a rákövetkezőkben pedig a modellek válaszait. Az adatok alatt összesítettem pár releváns adatot, többek között a pontosságot, kiegyensúlyozott pontosságot és az egyenesen és fordított sorrendben beadott mondatok egyezési arányát. Előbbiekhez beépített Google Táblázatok függvényeket használtam, míg utóbbihoz Apps Scriptet, a Táblázatok egy bővítményét, amellyel saját függvények definiálhatók és használhatók a táblázat bármely cellájában.

Az első sorban szerepelnek a gold standard értékek. A TP, FP, FN, TN címkék számát és eloszlását a Google Táblázatok beépített függvényei segítségével határoztam meg, és szimplán összeszámoltam a modellek által predikált "Yes" és "No" válaszok soronkénti megegyezését az azonos sorban szereplő gold standard értékekkel.

Egyes modelleknél (pl. Gemini 2.5 Pro Experimental 03-25) problémát okozott az, hogy nem csupán egy 'Yes', vagy egy 'No' válasszal tért vissza, hanem gondolatmenetet, magyarázatot is hozzáfűzött - akkor is, ha a promptban ennek az elhagyására kértem. Így

pl. az "I know! The answer is Yes." szöveg a Google Táblázatok függvényemben 'No'-ként lenne beleszámítva, akkor is, ha utána a modell szerint. Emiatt ezeknek a modellek outputjának a feldolgozását manuálisan végeztem.

A WiC saját kézi kiértékelése

A WiC honlapján az áll, hogy az emberi, manuális kiértékeléssel körülbelül 60%-ot tudtak elérni, azaz 80%-os pontossággal tudták a teszt adathalmazból vett, véletlenszerű példákról megállapítani, hogy a szó a 2 mondatban ugyanazt jelenti-e (a gold standard alapján). Ez nem túl magas, de érthető, tekintve, hogy ezek többsége ritka szó, és a mondatok nyelvezete is sokszor régies, vagy nagyon tudományos. A módszer az volt, hogy minden annotátor kapott 100 példát, és semmilyen segédeszközt nem használhattak.

Hogy meggyőződjek a nehézségről, én is eldöntöttem 60 bejegyzésről a gold standard ismerete nélkül, hogy a vizsgált szó ugyanazt jelenti-e a két mondatban. Az én eredményem 63.33%-os pontosság és 63.17%-os kiegyensúlyozott pontosság lett. Az eredményeim megtalálhatóak a kiértékelő rendszeremben a `manual_evaluation` mappában.

Futtatás Google Colab felületen

Ahhoz, hogy nagy nyelvi modell válaszait emberi időben megkapjuk, anélkül, hogy órákig felhasználnánk a személyi gépünk teljes erőforrását, GPU-n kell futtatni a modellt. Ez töredékére csökkenti a futási időt. Az általam elérhető eszközök egyike sem GPU-s. Erre kínált megoldást a Google Colab. Ez a platform segítségével távoli, Google szerverek gépjeit használhatjuk, nem kell a gépünk háttértárát, memóriáját stb. használni. Saját szkriptjeink futtatásához, és akár GPU-s futtatókörnyezetet is használhatunk, áthidalva a gépünk hardveres korlátait. Hátrányai, hogy a programunk változói, memóriája csak a session erejéig tárolódik, és lassabb futásidejű, mint a lokálisan futó program. A hátrányain úgy kerekedtem felül, hogy AutoClicker szoftvert állítottam a jegyzetfüzet egy véletlenszerű pontjára, így életben tartva az aktív munkamenetet akár órákig is.

3.1.8. Eredmények

Kiválasztottam 60 véletlenszerű kérdést a `test` halmazból. Az összes pontosság és kiegyensúlyozott pontosság kiértékelésem 50% és 60% közötti eredményt ért el úgy, hogy mindegyik modell pontosan ugyanazt a promptot kapta. Leggyengébben a Phi szerepelt normál sorrendben (53.33% és 56.03%), fordított sorrendben (58.33% és 60.71%). A Gemma normál sorrendben (60.00% és 61.83%), fordított sorrendben (55.00% és 56.47%), eredményt ért el, míg a QWEN egyenes sorrendben (56.67% és 57.37%), fordított sorrendben pedig szinte azonos pontszámot ért el mindkét metrikában (58.33% és 58.93%). A kiegyensúlyozott pontosság magasabb, mint az egyezés mind a három modell esetében, ami azt sugallja, hogy egyenlő mértékben hajlamosak igennel és nemmel válaszolni.

A függelékben található táblázatban összesítettem a három nyelvi modell teljesítményét különböző nehézségű és sorrendű kérdések (normál és fordított) esetén, és konzisztenciájukat is feltüntettem. Az értékelés pontosság, kiegyensúlyozott pontosság és

egyezés százalékban történt. A modellek viselkedése és erősségei/hátrányai a megjegyzések oszlopban is kiemelésre kerültek.[4.5]

3.1.9. A Phi erősségei

A Phi-4-mini-instruct modell könnyű kérdések esetén kiemelkedően magas, 89.47% pontosságot és 88.89% kiegyensúlyozott pontosságot ér el normál sorrendben. Ezenkívül a modell stabil és konzisztens teljesítményt nyújt, ahogy a megjegyzések is jelzik. A közepes nehézségű fordított kérdéseknél is viszonylag jó, 58.33% pontosságot és 60.71% kiegyensúlyozott pontosságot ért el.

	Accuracy	**Balanced Accuracy**
Könnyű N	89.47%	88.89%
Közepes F	58.33%	60.71%

3.2. táblázat. A Phi Legjobb Eredményei

3.1.10. A Phi gyengeségei

Bár a Phi-4-mini-instruct stabilnak mondható, a közepes nehézségű kérdéseknél a pontossága viszonylag alacsony (53.33% normál, 58.33% fordított sorrendben), annak ellenére, hogy az egyezés rendkívül magas (94.92%). Ez arra utal, hogy gyakran ugyanazt a hibát követi el, a szavak sorrendje nem befolyásolja őt különösebben a döntés meghozásában. Ez jó jel, hiszen azt jelenti, a modell képes a szavak sorrendjétől elrugaszkodni és absztraktabban, a feladat megoldására koncentrálva válaszolni. A nehéz kérdésekre pedig teljesen egyoldalúvá válik, és szinte minden idegen szó- és mondatkörnyezetet eltérő jelentésűnek kategorizál, tehát egyértelműen meghatározható ennek a modellnek a teljesítménykorlátja.

3.1.11. A Gemma erősségei

A gemma-2-2b-t modell a könnyű normál sorrendű kérdéseken a legmagasabb pontosságot érte el a három modell közül, 94.74% pontossággal és 95.00% kiegyensúlyozott pontossággal, ami kiváló nyelvérzékre utal. A közepes és nehéz kérdések esetén is 10%-kal felülmúlta a másik 2 modellt. Megfigyeltem egyébként, hogy a Gemma-2-2b-it helyes válaszai és hibái is nagyrészt megegyeznek a Gemma-2.5 válaszaival, ami bizonyítja az architekturális alapjuk hasonlóságát. A Gemma-2-2b-it a nehéz kérdésekre is 67%-os arányban jól válaszolt, jóval felülmúlva ezzel a két másik megvizsgált modellt. A Gemmának további előnye, hogy Colabos környezetben jóval gyorsabban fut, mint a Phi és a Qwen modellek, 1 perc alatt akár 60 kérdésre is képes válaszolni.

3.1.12. A Gemma gyengeségei

A gemma-2-2b-t modell a könnyű fordított kérdések esetén meglepően alacsony pontosságot mutat (63.16%), ami jelentős romlás a normál sorrendhez képest.

	Accuracy	Balanced Accuracy
Könnyű N	94.74%	95.00%
Közepes N	60.00%	61.83%
Nehéz N	68.75%	72.22%
Nehéz F	60.00%	55.56%

3.3. táblázat. A Gemma Legjobb Eredményei

3.1.13. A Qwen erősségei

A Qwen1.5-1.8B-Chat a Gemmánál jobb teljesítményt nyújtott egyszerű kérdéseken.

3.1.14. A Qwen gyengeségei

A Qwen1.5-1.8B-Chat a könnyű normál sorrendű kérdéseken gyengén teljesített a három modell közül, mindössze 68.42% pontossággal és nem logikus érvelésekkel. Ez arra utal, hogy az egyszerűbb feladatok megoldásában kevésbé hatékony, mint a többi modell. A nehéz kérdésekre a Qwen egyáltalán nem volt jó, és érdekes módon a Phi-vel ellentétesen, amely mindent a hamis kategóriába sorolt, a Qwen túlságosan megengedő és határozatlan volt, és a kérdések 84, majd 100%-ában egyezőnek feltételezte a szavak jelentését a két mondatban.

3.1.15. Összegzés

Összességében elmondható, hogy a Gemma teljesített a legjobban a három modell közül. Pontos, nem preferálja aránytalanul az egyik osztályt, öntudatos, figyel a saját korábbi válaszaira és konzisztens is, azaz el tud rugaszkodni a szavak sorrendjétől, absztraktabb módon a feladaton tartja a fókuszt. Általános célú feladatokra, például egy programozási feladat megoldására sokkal rosszabbul teljesít, mint egy GPT-4o, Claude 3.7 Sonnet, Grok vagy DeepSeek modelljei. A gemma-2-2b-it nem általános célú modell, hanem elsősorban utasításkövetésre lett optimalizálva.

4. fejezet

Saját rendszer fejlesztése - A részletes kimutatásokat tartalmazó megoldás

4.1. Technológiák

- NLTK WordNet – szóértelmezéshez
- SentenceTransformer – BERT-alapú mondatembeddinghez
- TF-IDF, cosine_similarity – szövegösszehasonlításhoz
- scikit-learn – küszöboptimalizálás, értékelés

Még mielőtt belekezdenék a tervezési gondolatmenetem kifejtésébe, hadd említsem meg, hogy miért ezeket a technológiákat használtam.

4.1.1. Az NLTK

Az NLTK (Natural Language Toolkit) [15] egy Python könyvtár, amely számos természetes nyelvfeldolgozási eszközt és adatkészletet kínál. Az NLTK kínál egyszerű eszközöket szójelentés-egyértelműsítésre, mint amilyen a korábban említett Lesk algoritmus implementációja, melyhez a WordNet [16] szolgáltatja a szükséges lexikális adatbázist.

A WordNet egy nagy lexikális adatbázis, amely angol nyelvű szavakat csoportosít úgynevezett "synset"-ekbe, amelyek kognitív szinonimákat és szavak közötti szemantikai kapcsolatokat is tartalmaznak. Minden synset egy fogalmi jelentést reprezentál, és a különböző synset-ek különböző jelentéseket fejeznek ki. A WordNet letöltését és elérhetőségének ellenőrzését a `download_wordnet_if_needed` függvénnyel automatizáltam. A szinonimák és jelentések kinyerésére a `get_synonyms` függvényt alkalmaztam, amely egy adott szóhoz tartozó összes lehetséges szinonimát visszaadja a WordNet segítségével. Az `expand_with_synonyms` függvény pedig egy teljes mondat szemantikai kiterjesztését végzi el, ami segítségével a mondatok közötti szemantikai átfedések explicit módon vizsgálhatóvá váltak.

A WiC modell működése

A modell egy előtanított nyelvi modellt használ, amely képes a szóhasználatok közötti finom eltéréseket azonosítani. A bemenet két mondatból és a vizsgált szóból áll, míg a kimenet egy bináris döntés: azonos vagy eltérő jelentés.

Adatfeldolgozás és annotáció

Az adatok előkészítése során szükség van megfelelő annotációs technikák alkalmazására. Az adatkészleteket standard WiC benchmarkokból nyerjük, amelyek kézi annotációval lettek ellenőrizve.

4.1.2. BERT

A BERT lényegében egy, a modellek reprezentációs képességét segítő szóbeágyazás, amely dinamikusan képes a mondatokhoz reprezentációt létrehozni. Ez nagy előny a régebbi, jól ismert statikus szóbeágyazásokkal szemben, mint például a Word2vec és a GloVe. Utóbbiak ugyanis egy szóhoz ugyanazt a szóvektort rendelik, bármilyen szövegkörnyezetben fordul is elő. A legnagyobb probléma ezzel, hogy sok szónak több, teljesen eltérő jelentése van, ám ezek a szóbeágyazások képtelenek modellezni szavak szemantikájának ezt a dinamikus természetét. A BERT ezzel szemben úgy hozza létre a mondatreprezentációt, hogy figyel a szövegkörnyezetre akár predikálás közben is. Ezzel lehetővé teszi, hogy ha egy szónak, vagy mondatnak több értelme is van, azt képes legyen a kontextus és a szó figyelembevételével megkülönböztetni.

A word2vec egy mára már elavultnak tekinthető technológia, ezért nem használtam, de a transzformerek megértéséhez hozzásegített.

Az utóbbi évek fejlődésével az olyan mélytanulási modellek, mint a BERT (Bidirectional Encoder Representations from Transformers) és más transzformátor-alapú nyelvi modellek még fejlettebb szövegreprezentációt kínálnak, figyelembe véve a kontextust és a szavak közötti összefüggéseket.

4.2. Tervezés

4.2.1. A kiértékelő rendszer célja

A készített alkalmazás célja az adatfeldolgozáson kívül, hogy egy olyan kiértékelő algoritmust hozzak létre, amely a Word-in-Context (WiC) feladatokhoz nyújt támogatást, lehetővé téve a szavak kontextusbeli jelentésének hatékony felismerését és osztályozását, továbbá minél nagyobb pontosságot ér el a WiC - The Word in Context Dataset teszthalmazon. A Google Colab jegyzetfüzetemhez hasonlóan itt is Pythont választottam, mert hatalmas a könyvtártámogatása NLP és mesterséges intelligencia területen, ami különösen alkalmassá teszi gyors fejlesztésre és adatfeldolgozásra, különösen szöveges adatokéra. Ezen kívül egy könnyen olvasható, általános célú, egyszerű szintaxisú, magas szintű programozási nyelv. Fejlesztői környezetnek a PyCharmot használtam.

Célom volt, hogy a kiértékelő rendszerem almappái egy önálló modulként használhatóak és futtathatóak legyenek - a környezet megfelelő felállítása után.

Fő szempontjaim voltak, hogy a projekt fájlstruktúrája könnyen értelmezhető legyen, kövesse a konvenciókat, könnyen lehessen benne tájékozódni, és logikusan legyen felépítve. Emellett a Python szkriptek, amennyire csak lehet, ne függjenek egymástól, tehát önállóan futtathatók legyenek. Természetesen ettől indokolt esetben eltértem, például amikor nagyméretű adatokat tároltam egy fájlban, azt másik rövid szkriptekben használtam fel, a könnyebb átláthatóság kedvéért.

Fájlstruktúra

A projektemben megtalálhatóak a Python projektekhez elengedhetetlen .venv (Virtuális Python környezet), requirements.txt a dependency-k követéséhez, Git fájlok (.gitignore, README, .git), és a fejlesztői környezet fájljai (.idea). Ezek nem érdekesek most számunkra. Az azonosítók elnevezését is igyekeztem rendkívül konzisztensen végezni: minden külső használatra szánt, úgynevezett "utility" Python szkriptem a `wic` előtaggal kezdődött. Amelyik pedig belső használatra volt szánva, vagy szabványos neve van (pl. __init__.py, setup.py), az nem kapott `wic` előtagot. Az ötletet többek között az Intel által fejlesztett OpenCV `cv2` digitális képfeldolgozásra szolgáló függvénykönyvtárából merítettem, amely többek között Python nyelven is használható. Itt ugyanis minden beimportálható függvény, enum, változó, konstans stb. a `cv2.` előtaggal kezdődik, amely rendkívül megkönnyíti a `cv2` azonosítóinak (Függvény-, Osztály-, Változó-, Konstans-, Metódus-, Paraméter-, Enum- és Névtérnevek) elkülönítését a saját azonosítók, és ezáltal elősegíti a kód olvashatóságát.

"Pythonic" alapelvek

A projektemben igyekeztem követni a Zen of Pythonban [17] megfogalmazott, "Pythonic" alapelveket és a Python közösség által elfogadott jó gyakorlatokat. A Pythonic elveket úgy alkalmaztam, hogy minél több információt expliciten adtam meg, funkcionálisan programoztam, kerülve a beágyazott blokkokat, és elősegítve a kód önmagát dokumentáló jellegét. A kódomban előforduló azonosítóknak, főleg a fájloknak, változóknak, függvényeknek rendszerint hosszú, *barokkosnak* nevezhető nevet adtam, hogy a fájl megnyitása nélkül is világos legyen, hogy mit csinál, miért felelős. A rendkívül hosszú azonosítónevek adása egyébként a nagyvállalati kultúrában, a Java és a legtöbb, objektumorientált nyelvet használó programozók körében is bevált konvenció, hiszen többek között csökkenti a dokumentáció szükségességét. Az összetett logikát egyszerűbb, egymás után következő lépésekre bontottam le, még ha ez kódismétléssel is járt, a könnyebb olvashatóság érdekében.

A projekt felépítése

A tervezés során számos lépést megtettem, hogy a projektem könnyen átlátható legyen:

- Mivel a szkriptjeim nagy részében több ezer soros fájlok adataival dolgoztam, és nem lett volna célszerű minden fájlba közvetlenül importálni ezt a nagy adatmennyiséget, ezen kívül környezettől függetlenül futtathatóvá akartam tenni a projektem, ezért automatizáltam az importálási folyamatot, és változókban mentettem el a fájlneveket és az elérési útvonalakat, amely szinte az összes (nem csak függvényeket tartalmazó) szkriptemben állítható az `actual_working_dataset` változóval.

```
# Define which dataset you want to work with
actual_working_dataset = 'test' # pick one from ['train', 'dev', 'test']
try:
    base_path = f"{PATH}/{actual_working_dataset}"
    data_file = os.path.normpath(os.path.join(base_path, f"{actual_working_dataset}.data.txt"))
    gold_file = os.path.normpath(os.path.join(base_path, f"{actual_working_dataset}.gold.txt"))
except FileNotFoundError:
    print(f"Warning: {actual_working_dataset} dataset not found. "
          "Please ensure it is available in the working directory.")
    print("You can download the WiC dataset from: https://pilehvar.github.io/wic/")
```

4.1. ábra. Az adathalmaz kiválasztásának automatizálásának módszere

- A harmadik féltől származó könyvtárak, csomagok és egyéb függőségek egységesen, `requirements.txt`-ben lettek definiálva. Ez egy Pythonos konvenció, amelyet én is követtem. A PyCharm eszközt kínál a projekt függőségeinek automatikus összegyűjtésére, és arra is, hogy a felhasználó egy gombnyomásra képes legyen ezeket egyszerre letölteni.

- **Modularitás:** A projektemben a lehető legkevesebb fájlokon átívelő függőséget alkalmaztam, tehát a projekt kódja annyira moduláris, amennyire csak lehet. Ez különösen hasznos a projekt kódjának refaktorálásakor, illetve akkor, ha csak egy, vagy pár fájlt kellett használnom a projektből egy bizonyos feladatra, mert biztosította a program kis méretét és gyorsaságát.

- Kivétel erre a nem importálásra szánt fájljaim, ahol ténylegesen történik a függvények meghívása a saját függvénykönyvtár fájljaimból, és a konfigurációs beállítások (pl. Elérési utak) tárolására szolgáló fájljaim.

Figyeltem a SOLID objektum-orientált programozási alapelvekre [18] is, főleg az Open-Closed alapelvre, hogy az alkalmazásom könnyen bővíthető, de nehezen elrontható legyen. A függvényeimet úgy paramétereztem, hogy úgy lehessen a kiértékelés módját változtatni, hogy csak 1 paramétert kell átírnunk hozzá, ezzel sok kódduplikálást megelőzve. Példák: A `compute_sentence_similarity` alapértelmezett módon a gyengébb, nagyon gyors TF-IDF vektorizáló módban, azonban a `mode="bert"` paraméterezéssel a nagyon lassú, de sokkal jobb teljesítményű SenseBERT beállítással fut le, amely az említett előtanított nyelvi modellt használja, amelyet kifejezetten a szóérzékelés (word sense disambiguation, WSD) feladatra optimalizáltak.

Másik példám, ahogy az `evaluate` függvény alapértelmezetten a hibahatár threshold kísérletező paraméterével végigpróbálja az összes temperature-t, és automatikusan a

legjobb eredményű kiértékelést választja. Ez hatékony, de nagyon sok időt vesz igénybe, ezért a paraméternek explicit értéket lehet adni. Ekkor csak erre az egy értékre fut le, ami gondos kiválasztást igényel a fejlesztő részéről, de nagyon gyors. Még egy példa, hogy az eredmény-kiíró függvényben meg lehet adni bőbeszédű és normál opciókat, amellyel kiválaszthatjuk, hogy mennyi üzenet jelenjen meg a konzolon, és az elfogadható bizonytalanság mértékét is lehet állítani. Ugyanitt egyesével is tudjuk állítani a paraméterek kiíratását.

Kommentek

Angol nyelvű projekt lévén a kommenteket egységesen angolul írtam mindenhol, ez segítette az internetes segítségkérést, például fórumokon.

4.3. Implementáció

Első lépésként inicializáltam a projektemet lokálisan és a GitHub verziókövető platformon. A projektemnek a "Peternity" nevet adtam. A "Peternity" egy jól hangzó és egyedi név, nincsen különösebb oka a névadásnak. A program fejlesztését inkrementális modellel végeztem, tehát kezdetben csak vázlatosan határoztam meg a kiértékelő rendszer funkcióit. Ennek a modellnek az előnye például a vízesés modellel szemben az, hogy a követelmények nincsenek konkrétan lerögzítve a fejlesztés elején, hanem folyamatosan lehet rajtuk változtatni. Ha eszembe jut egy újabb módszer, amivel jobb eredményt lehet elérni, vagy egy funkció, amely növeli a program érthetőségét, átláthatóságát, akkor nem kell újratervezni a programot.

Mappák

- **independent_scripts mappa**
  Az önállóan működő, független Python szkripteket ide raktam.

- **llm_prompts mappa**
  Az llm_prompts mappában lévő szkriptekkel olyan promptokat hoztam létre, amelyektől jó eredményt lehet elvárni a nyelvi modelleken való promptolás esetén. Emiatt nyelvtanilag helyesek, a tokenek egymástól egyértelműen elkülönülnek, és szerkezetileg megfelelnek a nyelvi modellek elvárásainak: a prompt elején van az utasítás, majd a kérdések következnek jól elkülönülve. A validáció és tesztelés érdekében készítettem pár nagyon egyszerű, és pár nagyon nehéz kérdést is a nagyon gyenge, illetve nagyon jól teljesítő modellek számára.

- **modules_and_data mappa**
  Ez a mappa a projekt futtatásához szükséges modulokat és az adathalmazokat tartalmazza. 2 részre oszlik:

`modules`: Ide tartoznak az adathalmaz tisztítását, felesleges, vagy használhatatlan adatok kiszűrését és egyéb adattranszformációkat végző Python szkriptjeim.

`data`: Az adatok előfeldolgozása és átalakítása ebben a mappában történik, például tokenizálás, normalizálás vagy TF-IDF számítás.

- **solution mappa**
  Szintén 2 részre oszlik
  `Implementation`: Ez a mappa tartalmazza a megoldásom végső implementációját és a projekt által generált eredményeket. Itt található a BERT-alapú megoldásom, valamint a generatív nyelvi modellek teljesítményének összehasonlítása.

  `Results`: Az itt található fájlok a különböző modellek által generált válaszokat, tévesztési mátrixokat, statisztikákat és egyéb mérőszámokat tartalmazzál

  Ez az önálló, fájlokra szétbontott, TF-IDF-alapú megoldás a `main.py` szkripttel indítható. Ez a kiértékelő szkript a pontosság mellett kiszámítja és megjeleníti a precizitást, fedést és F1 pontszámot, valamint vizuálisan ábrázolja a tévesztési mátrixot.

- **use_modell_locally mappa**
  A `py` almappában webes API-on keresztül tesztelhetők az OpenAI és az Anthropic modelljei. A kód a fejlesztő cégek hivatalos modell dokumentációjából erednek. Az első szkripttel GPT modellek hívhatóak meg az OpenAI API-n keresztül az OpenAI dokumentáció [19] által ajánlott módon egy adott szó jelentésének két mondatbeli összehasonlítására, míg a második fájlban ugyanilyen módon a Claude modelljei használhatók az Anthropic API segítségével az Anthropic dokumentáció szerint [20].

- **WiC_dataset mappa**
  A WiC (Word-in-Context) adathalmaz itt található

- **demo_files** mappa
  A **demo_files** mappában található egy alternatív kompaktabb megoldás, amely csak a pontosságot mutatja ki, de cserébe sokkal konfigurálhatóbb.
  A `wic_tfidf_or_sensebert_baseline_single.py`
  szkript egy alap Word-in-Context (WiC) probléma megoldására szolgáló baseline rendszert valósít meg, amely szóértelem-felismerésre (Word Sense Disambiguation, WSD) és mondat-hasonlóság számítására épít. Ez egy konfigurálható fájl, futtatásával a kiválasztott adathalmaz (train, dev vagy test) beolvasásra kerül, TF-IDF vagy SenseBert módban, opcionális kiírás-részletesség, küszöbérték- és szinonimaoptimalizálás beállításokkal. A WordNet segítségével meghatározza a cél szó legrelevánsabb jelentését az adott mondat alapján, és csak a kontextushoz illő szinonimákat adja hozzá a mondathoz. TF-IDF és/vagy BERT-alapú mondatembeddingből egy hibrid hasonlósági értéket számol. Ezután meghatározza az optimális határértéket a "T"/"F" címkék elválasztásához, végül összehasonlítja a predikciókat a gold standard címkékkel, opcionálisan figyelembe véve egy bizonytalansági zónát. A legjobb eredményt a SenseBERT beállítással, legjobb threshold mellett, 19 perc futással értem el, mind a 3 halmazon 60%-nál jobb eredményt értem el.

Adathalmaz	Mód	Pontosság (%)	Futásidő (másodperc)
train	BERT	59.912	8294.84
train	TF-IDF	57.970	5903.44
dev	BERT	60.696	128.44
dev	TF-IDF	62.088	91.95
test	BERT	59.571	382.36
test	TF-IDF	60.658	332.65

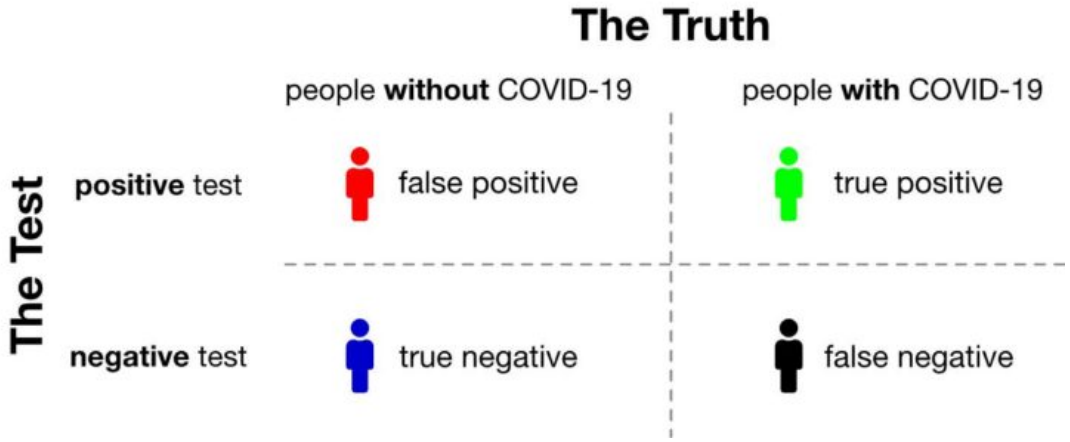
4.1. táblázat. A `wic_tfidf_or_sensebert_baseline_single.py` eredményei és futásideje a különböző beállítások mellett. Érdekes módon az egyszerűbb kérdéseket tartalmazó `train` halmazon a BERT jobban teljesít, míg a nehezebb kérdésekből álló `dev` és `test` esetén a TF-IDF minimálisan pontosabb. Ez alapján leszűrhető, hogy a BERT módszer egyszerűbb kérdések esetén hasznosabb, főleg ha nem lényeges a gyors a futásidő.

4.4. Tesztelés és eredmény

4.4.1. Kiértékelés

Klasszifikációs módszer

Az algoritmusaim teljesítményét egy tévesztési mátrix segítségével, klasszifikációs módszerrel értékeltem ki, ami például Covid-teszt eredményekkel jól szemléltethető. Covid esetén pozitív a teszt, ha a páciens beteg, negatív, ha egészséges. Továbbá igaz a predikció, ha helyesen ítéltük meg az egészségi állapotát, és hamis, ha helytelenül ítéltük meg. Ezeknek a kombinálásával kapjuk meg a TN, FP, TN, FN értékeket.



4.2. ábra. Covid-19 példa a TP, TN, FP, FN alkalmazására

A WiC feladat esetében ezt úgy kell értelmeznünk, hogy pozitív a gold standard, ha a szó ugyanazt jelenti mind a két mondatban, negatív, ha eltérő dolgot. Igaz, ha helyesen ítélte meg a modell a szó jelentését és hamis, ha helytelenül ítélte meg. A modell

válaszának a gold standarddal való összevetésekor minden esetet az alábbi négy kategória egyikébe sorolhatjuk:

- **True Positive (TP):** A modell helyesen válaszol Yes-t, amikor a gold címke T, azaz valóban azonos a szó jelentése.
- **False Positive (FP):** A modell tévesen válaszol Yes-t, miközben a gold címke F, tehát eltér a szó jelentése.
- **False Negative (FN):** A modell tévesen válaszol No-t, miközben a gold címke T, tehát azonos lenne a szó jelentése.
- **True Negative (TN):** A modell helyesen válaszol No-t, amikor a gold címke F, azaz valóban különböző a szó jelentése.

$$\begin{bmatrix} \text{Igaz pozitív (TP)} & \text{Hamis pozitív (FP)} \\ \text{Hamis negatív (FN)} & \text{Igaz negatív (TN)} \end{bmatrix} \tag{4.1}$$

Az egyfájlos módszerem eredményei a 4.3. ábrán láthatók. 4.5 A kiértékelő rendszerem elérte a célját, hogy gyorsan ki tud értékelni több ezer kérdést, és legalább 60%-os pontossággal el tudja dönteni a szavakról, hogy ugyanazt jelentik-e a 2 mondatban, vagy sem.

Tanulságok, hipotézisek

A WiC feladaton a természetes nyelvek szavainak szemantikájának komplexitásából adódóan gyakorlatilag lehetetlen 100%-os pontosságot elérni, de ez nem is cél. Kézi, emberi kiértékeléssel 80%-os eredményt lehetett elérni, a legjobb rendszerek pedig 70-72%-ot értek el. A cél, hogy egy olyan WiC adathalmaz-kiértékelő szoftver készüljön, amely hosszú távon segít az embereknek, könnyen integrálható más projektekbe és széles skálán terjeszthető.

A kvantált modellek, mint a Gemma-2-2b-it, kisebb méretük miatt könnyebben futtathatók korlátozott hardveres erőforrásokkal, de ez a méretcsökkenés a teljesítmény rovására megy (pl. lassabb válaszidő, pontatlanabb eredmények).

A Gemma modell bizonyos specifikus feladatokra (pl. Word-in-Context) megfelelően alkalmazható, de általános célú kérdések esetén jelentősen alulmúlhatja a nagyobb modelleket, mint pl. GPT-4o vagy Claude.

A kis modellek érzékenyebbek a bemeneti prompt formátumára, mint a nagyok, különösen a sorrendiséget tekintve. Ha egy modellt ugyanarra a kérdésre eltérő sorrendű mondatokkal kérdezünk meg, akkor a válaszokban jelentős eltérések lehetnek. Emiatt az egyenként feltett kérdések pontosabb eredményt adnak, mint a több kérdést egyben tartalmazó prompt.

A nyílt forráskódú és offline futtatható modellek előnye, hogy teljes adatvédelmi kontrollt biztosítanak, mivel nem kommunikálnak a külvilággal, így érzékeny adatok feldolgozására biztonságosabbak.

4.4.2. Tanulságok, hipotézisek

A WiC feladaton a természetes nyelvek szavainak szemantikájának komplexitásából adódóan gyakorlatilag lehetetlen 100%-os pontosságot elérni, de ez nem is cél. Kézi kiértékeléssel 80%-os eredményt lehetett elérni, a legjobb rendszerek pedig 70-72%-ot értek el. A cél egy olyan WiC adathalmaz-kiértékelő szoftver, mely a jövőben felhasználható hasonló NLP vagy adatfeldolgozási feladatokra, könnyen integrálható más projektekbe és széles skálán terjeszthető. Emellett persze a személyes célom volt a Python ismereteim fejlesztése.

A kvantált modellek, mint a gemma-2-2b-it-GGUF, kisebb méretük miatt könnyebben futtathatók korlátozott hardveres erőforrásokkal, de ez a méretcsökkenés a teljesítmény rovására megy (pl. lassabb válaszidő, pontatlanabb eredmények).

A Gemma modell bizonyos specifikus feladatokra (pl. Word-in-Context) megfelelően alkalmazható, de általános célú kérdések esetén jelentősen alulmúlja a nagyobb modelleket, mint pl. GPT-4o vagy Claude.

A kis modellek érzékenyebbek a bemeneti prompt formátumára, mint a nagyok, különösen a sorrendiséget tekintve. Ha egy modellt ugyanarra a kérdésre eltérő sorrendű mondatokkal kérdezünk meg, akkor a válaszokban jelentős eltérések lehetnek. Emiatt az egyenként feltett kérdések pontosabb eredményt adnak, mint a több kérdést egyben tartalmazó prompt.

A nyílt forráskódú és offline futtatható modellek előnye, hogy teljes adatvédelmi kontrollt biztosítanak, mivel nem kommunikálnak a külvilággal, így érzékeny adatok feldolgozására biztonságosabbak.

4.4.3. Tervek a jövőre nézve

Jövőbeli tervem ez a projekt után, hogy a XL-WiC [21] és a WiC-TSV [22] problémát is mélyebben megismerjem, és lehetőleg magyar nyelvre is hasonló adathalmazokat találjak, és hasonló rendszert hozzak létre. Ezen kívül érdekes kísérlet lenne az, hogy a különböző méretű és tulajdonságú modellek többkörös beszélgetésben (multi-turn conversation) mennyire képesek következetesek maradni és önmaguknak nem ellentmondani.

4.5. Összegzés

Úgy gondolom, sikerült átfogó elemzést nyújtanom a különböző LLM-ek WiC feladatokban nyújtott teljesítményéről 2025 tavaszára vonatkozóan. Fontosnak tartom megjegyezni, hogy ez egy rendkívül gyorsan fejlődő iparág, így amit a szakdolgozatban állítok, az a jövőbeli olvasónak elévülhet. A dolgozatomat úgy állítottam össze, hogy nyomon követhető és reprodukálható legyen, ezért nyílt forráskódú eszközökre (pl. Hugging Face, NLTK, BERT) és nyilvános adathalmazokra támaszkodtam.

Eredmények, ábrák

Modell	Nehézség	Pontosság (%)	Kiegy. Pontosság (%)	Egyezés (%)	Megjegyzés
Phi	Könnyű N	89.47	88.89	89.43	Konzisztensek a válaszai
	Könnyű F	78.95	77.78		
	Közepes N	53.33	56.03	95.00	Legjobb egyezés, de alacsony pontosság
	Közepes F	58.33	60.71		
	Nehéz N	53.33	44.44	60.00	Válasza szinte minden kérdésre "nem"
	Nehéz F	60.00	52.78		
Gemma	Könnyű N	94.74	95.00	68.42	Legpontosabb könnyűekre
	Könnyű F	63.16	65.00		
	Közepes N	60.00	61.83	85.00	Legpontosabb közepesekre
	Közepes F	55.00	56.47		
	Nehéz N	68.75	72.22	73.33	Legjobb nehéz kérdéseken is
	Nehéz F	60.00	55.56		
Qwen	Könnyű N	68.42	69.44	74.00	Leggyengébb az egyszerű kérdéseken
	Könnyű F	73.68	74.44		
	Közepes N	56.67	57.37	91.67	Erős egyezés
	Közepes F	58.33	58.93		
	Nehéz N	46.67	52.78	80.00	Válasza szinte minden kérdésre "igen"
	Nehéz F	40.00	50.00		

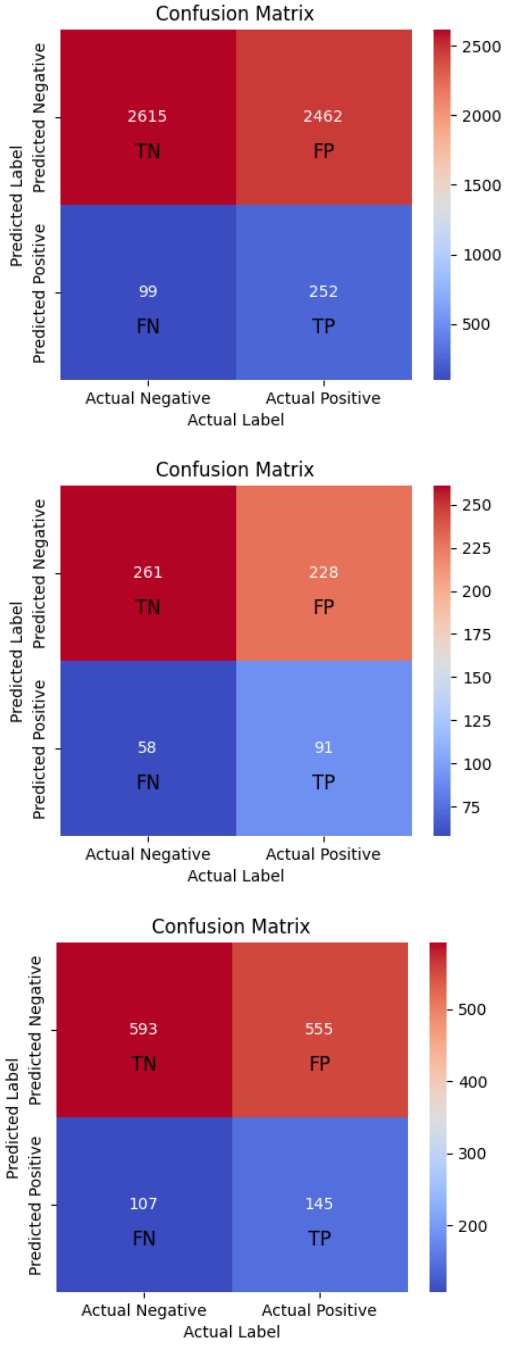
4.2. táblázat. A három nyelvi modell teljesítménye különböző kérdésnehézségek és sorrendek mentén. N=normál, F=fordított (pontosság, kiegyensúlyozott pontosság és egyezés %)

```
# Train dataset
Evaluation Metrics:
Accuracy: 52.819%
Precision: 71.795%
Recall: 9.285%
F1 Score: 16.444%
Correct predictions: 2867/5428


# Dev dataset
Evaluation Metrics:
Accuracy: 55.172%
Precision: 61.074%
Recall: 28.527%
F1 Score: 38.889%
Correct predictions: 352/638


# Test dataset
Evaluation Metrics:
Accuracy: 52.714%
Precision: 57.540%
Recall: 20.714%
F1 Score: 30.462%
Correct predictions: 738/1400
```

4.3. ábra. A `solution` eredményei a train, dev és test halmazra



4.4. ábra. Az egyfájlos megoldás tévesztési mátrixai a train, dev és test halmazra

Példa a A sentence_normalizer használatára

```
 # sentence_normalizer.py
 test_sentence = (
  r"My boss is out on another of his three martini lunches ."
  r" Will you join us at six o'clock for martinis ? We 've
been swimming for hours just to get to the other side ."
  r" A big fish was swimming in the tank . Do n't fire until
you see the whites of their eyes . The gun fired ."
  )

 print("Using NEW algorithm (default):")
 print(make_sentence_human_readable(test_sentence))
 # Output:
 # My boss is out on another of his three martini lunches.
Will you join us at six o\'clock for martinis?
 # We\'ve been swimming for hours just to get to the other
side. A big fish was swimming in the tank.
 # Don\'t fire until you see the whites of their eyes. The
gun fired.
```

Irodalom

[1] 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)*. Szerk. Jill Burstein, Christy Doran és Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. jún., 1267–1273. old. DOI: 10.18653/v1/N19-1128. URL: https://aclanthology.org/N19-1128/.

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

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

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

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

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

[7] The Verge. *The New York Times prohibits using its content to train AI models*. 2023. URL: https://www.theverge.com/2023/8/14/23831109/the-new-york-times-ai-web-scraping-rules-terms-of-service.

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

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

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

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

[12] Microsoft. *Phi-4-mini-instruct*. https://huggingface.co/microsoft/Phi-4-mini-instruct. 2023.

[13] Google. *Gemma-2-2b-it*. https://huggingface.co/google/gemma-2-2b-it. 2023.

[14] Qwen. *Qwen1.5-1.8B-Chat*. https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat. 2023.

[15] Steven Bird és Edward Loper. „NLTK: The Natural Language Toolkit". *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Barcelona, Spain: Association for Computational Linguistics, 2004. júl., 214–217. old. URL: https://aclanthology.org/P04-3031/.

[16] George A. Miller. „WordNet: A Lexical Database for English". *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*. 1994. URL: https://aclanthology.org/H94-1111/.

[17] Tim Peters. *PEP 20 – The Zen of Python*. https://peps.python.org/pep-0020/. Python Enhancement Proposal. 2004.

[18] Samuel Oloruntoba és Anish Singh Walia. *SOLID: The First 5 Principles of Object Oriented Design*. 2024. ápr. URL: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.

[19] OpenAI. *OpenAI API Documentation Overview*. 2025. URL: https://platform.openai.com/docs/overview.

[20] Anthropic. *Client SDKs - Anthropic API*. https://docs.anthropic.com/en/api/client-sdks. 2025.

[21] Alessandro Raganato és tsai. *XL-WiC: A Multilingual Benchmark for Evaluating Semantic Contextualization*. 2020. arXiv: 2010.06478 [cs.CL]. URL: https://arxiv.org/abs/2010.06478.

[22] Anna Breit és tsai. *WiC-TSV: An Evaluation Benchmark for Target Sense Verification of Words in Context*. 2021. arXiv: 2004.15016 [cs.CL]. URL: https://arxiv.org/abs/2004.15016.

Nyilatkozat

Alulírott Fábián Bernát, programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszékén készítettem, 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. május 24. .....

aláírás

Köszönetnyilvánítás

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

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

Elektronikus mellékletek

- A modelleket futtató Google Colab jegyzetfüzetem: ezen a linken található.
- A kiértékelő rendszer és függvénykönyvtár GitHubról a https://github.com/Fabbernat/Peternity repozitóriumból tölthető le.
- A nyelvi modellek tesztelése és kiértékelése a Generative Language Models táblázatban tekinthető meg.