Gradus Vol 10, No 2 (2023)
ISSN 2064-8014

MESTERSÉGES INTELLIGENCIA ADATKÉSZLETEK HATÉKONY FORDÍTÁSA NYÍLT FORRÁSÚ TECHNOLÓGIÁK SEGÍTSÉGÉVEL

EFFICIENT TRANSLATION OF ARTIFICIAL INTELLIGENCE DATASETS WITH OPEN-SOURCE TECHNOLOGIES

Cserkó József [0009-0001-3271-9891*], *Pásztor Attila* [0000-0001-7354-5114]

Informatika Tanszék, GAMF Műszaki és Informatikai Kar, Neumann János Egyetem, Magyarország
https://doi.org/10.47833/2023.2.CSC.021

Kulcsszavak:
Mesterséges Intelligencia,
MI,
adatkészlet,
fordítás

Keywords:
Artificial Intelligence,
AI,
datasets,
translation

Cikktörténet:
Beérkezett 2023. szept. 11.
Átdolgozva 2023. nov. 15.
Elfogadva 2023. nov. 17.

Összefoglalás

A Mesterséges Intelligencia (MI) egyre inkább megkerülhetetlen technológiává válik az utóbbi években és forradalmasítja az ipar, az oktatás és a magánélet különböző területeit. Ahhoz azonban, hogy megfelelő minőségű adatmodelleket készíthessünk, igen nagy mennyiségű adatra van szükség. Csak a megfelelő minőségű adatforrások birtokában lehet jó minőségű alkalmazásokat fejleszteni és kiaknázni az MI-ben rejlő teljes potenciált. Ezek az adatforrások használhatóak fel a modellek létrehozása és betanítása során, viszont igen nehéz jó minőségűeke találni.

Jelentős probléma az, hogy a legtöbb adatforrás vagy adatkészlet angol nyelven érhető el.

A cikk összefoglalja egy saját környezetben megvalósított fordító alkalmazás elkészítését. Vizsgálja az egyes architektúrák hatékonyságát és bemutatja a fejlesztési folyamat lépéseit és kihívásait is.

Abstract

Artificial Intelligence (AI) has become an increasingly inescapable technology in recent years and is revolutionising different areas of industry, education and private life. However, in order to build data models of the right quality, a very large amount of data is needed. Only with the right quality data sources can high-quality applications be developed and the full potential of AI be realised. These data sources can be used to create and train models, but it is very difficult to find good quality ones.

A significant problem is that most data sources or data sets are available in English.

This article summarises the development of a compiler application in a proprietary environment. It examines the effectiveness of the different architectures and describes the steps and challenges of the development process.

* Kapcsolattartó szerző.
E-mail cím: cserko.jozsef@nje.hu

1. Bevezetés

Számos tudós szerint a mesterséges intelligencia a legfontosabb technológia napjainkban. Véleményük szerint, ez az új ipari forradalom, ami teljesen meg fogja változtatni a hétköznapi életet [1]. Ez bizonyos mértékeben igaz lehet, az egyetemek és az oktatásban tevékenykedők azonban rámutatnak ennek a technológiának a gyengeségeire, különösen az oktatás területén.

Az akadémiai szférában egyesek tiltanák, mások szabaddá tennék az MI használatát a diákok számára. [2]. Az a tapasztalat, hogy nem érdemes a teljes tiltás irányába menni, mivel az oktatók nem rendelkeznek felügyeleti joggal a tanulók eszközei felett [3] [4].

Mivel az egyetemek látják, hogy nem tudják megakadályozni a nyelvi modellek használatát, több új projekt is indult, amelyek használni kezdték ezeket az oktatásban [5].

Egyetemünk is egy új projektbe kezdett, hogy elkészítse a hallgatók támogatására szolgáló mesterséges intelligencia modellünket. A cél az, hogy témakörönként külön modelleket hozzunk létre, amelyek támogatják az informatika, a mérnöki tudományok, vagy a logisztika elsajátítását [9] [10] [11] [12] [13] 14].

Emellett az MI modellek használata elősegíti a távoktatás fejlesztését is, amely előtérbe került a koronavírus járvány megjelenése után [15].

Az első nehézség az, hogy mindegyikhez megfelelő forrásokat találjunk. Az interneten számos jó forrás található, de sajnos ezek többnyire angol nyelvűek. Emellett, a hatalmas adatforrások fordítása drága és időigényes lehet, és robusztus hardvert igényel.

Ez a cikk lépésről lépésre ismerteti az angol adatforrások más nyelvekre történő hatékony fordításának lehetőségét speciális hardver- és szoftverkörnyezetek segítségével. Megvizsgálja a különböző beállítások és felhasználási esetek közötti különbségeket.

2. Az Adatkészletek

Amikor adatkészletekről beszélünk, azok általában valamilyen program által az interneten gyűjtött adatokból állnak. Ezek nagyon különböző minőségűek lehetnek. Szinte minden esetben szükséges ellenőrizni és megfelelő formára hozni őket, mielőtt a Mesterséges Intelligencia rendszer tanítása megkezdődik.

Az adatok egységes formára hozását normalizálásnak nevezik. Ez egyszerű műveletekből áll, például a “padding” vagy a “clipping” technika. Az első esetben egy bizonyos méretre egészítik ki a szöveges adatokat, a másodikban pedig a túl hosszú adatsorokat rövidítik le.

Elmondható, hogy szinte minden esetben szükség van normalizációra vagy az adatok elsődleges feldolgozására. Az, hogy milyen technikát alkalmazunk, mindig az alap adatkészlet minőségétől függ.

Ha a megfelelő adatkészlet kiválasztásra került, meg kell vizsgálnunk, hogy a használni kívánt modell milyen bemeneti adatokat fogad el. Például mi a maximálisan megengedett hosszúságú szöveg, amit egy lépésben képes feldolgozni, vagy milyennek kell lennie a szöveg kódolásának? Esetleg vannak-e speciális követelmények? Tehát nem használhatjuk fel azonnal a kapott adatkészletet, a modellünk tanítása előtt át kell alakítanunk.

Ha felügyelt tanulást alkalmazunk, akkor nem elég egyszerű szöveges listákkal dolgozni, hanem valamilyen strukturált szövegre van szükség, ami a legtöbbször valamilyen lista, amely kulcs-érték párokat tartalmaz. Ha Python program segítségével strukturálták az adatokat, akkor könyvtárak listájáról van szó. Magát az adatkészletet xml vagy json formátumban tárolják, ezek közül a json formátum az elterjedtebb, mivel kisebb a fájlméret és rugalmasabb az adatok olvasása. A legtöbb programnyelv biztosít rutinokat a json állományokkal való munkához, természetesen a Python is. Nagyon egyszerű a json állományok írása és olvasása, és a tárolt adatok feldolgozása Python nyelven.

Nem felügyelt tanítás esetén nem kell, hogy strukturált legyen a szöveg, viszont itt nehezebb is a modell működésének az ellenőrzése. Ebben az esetben a modell maga talál rá a szöveg összefüggéseire és az ellenőrzés leginkább a modell tanítása után lehetséges, nem a tanítási folyamat közben.

A legnépszerűbb oldalak, ahol adatkészletek találhatók, a Kaggle vagy a Huggingface. Mindkettőn nagy a választék, de a Huggingface jobb támogatást ad, ha valaki tovább szeretné fejleszteni az egyes szetteket.

3. A Fordító Modell Kiválasztása

Lehetőség van a saját gépen vagy felhőszolgáltatást használva is futtatni a kiválasztott modelleket.

Az alábbi részben összehasonlítjuk a különböző platformok főbb előnyeit és hátrányait.

Bérelt felhő alapú megoldás esetén nem kell beruházni a költséges hardverekbe. Emellett csak a használat alapján kell fizetni, az elérhető csomagok pedig igen sokfélék. Egészen erős szolgáltatást is lehet bérelni, ahol nagyságrendekkel gyorsabban futnak a modellek, viszont ezek természetesen drágábbak is. Itt nem kell az üzemeltetéssel, karbantartással sem törődnünk. Hátránya viszont, hogy nem költséghatékony akkor, ha előre nem tudjuk pontosan, hogy mennyi időt fog igénybe venni a feladat [16]. Például a jelen projekten is sok teszt futtatásra volt szükség amíg megfelelően le tudtuk tesztelni, hogy milyen beállításokkal, milyen előkészített adatkészlettel kapjuk a legjobb eredményt.

A saját hardver előnye, hogy nem ragadós. Ezt a kifejezést arra használják, hogy az egyes felhőszolgáltatások eltérő architektúrát használnak. Ha egy alkalmazást a Google, Microsoft vagy az Amazon rendszerében optimalizálunk, akkor nagy valószínűséggel nem fog jól skálázódni a konkurencia hardverén. Tehát könnyű beragadni egy adott környezetbe. Viszont a saját hardver esetén tagadhatatlan hátrány a nagy beruházási költség.

A teszteket egy saját szerveren futtattuk, amely a következő fontosabb hardverelemeket tartalmazta:

- CPU: AMD Ryzen 1900x (8 mag, 16 szál)
- RAM: 32GB DDR4
- GPU: Nvidia RTX 3090 24GB
- PSU: 1000 W

Mivel a gép új és használt alkatrészeket is tartalmazott, így a becsült bekerülési költsége 700.000 Ft volt 2023-ban.

Összehasonlítva az Amazon SageMaker platformjával, ez ott körülbelül 350 Ft ($1) óránként. Tehát a saját hardver 2000-szer drágább. Ez alapján úgy tűnik, hogy sokkal jobban megéri a felhőszoltáltatás. Viszont az Amazon külön díjat számít fel a tárhelyért is, így valamivel nagyobb összeget kell fizetni. Emellett, ha azzal kalkulálunk, hogy 4 óra a teljes idő, amit a fordítás vagy más modell futtatása igényel, könnyen hibát követhetünk el. Tapasztalataink szerint a megfelelő optimalizációk és tesztek könnyen 10-30-szor annyi ideig is tartanak, mint a konkrét munkafolyamat. Ez abból fakad, hogy a használt Python kódban is lehetnek hibák, hálózati problémák is felléphetnek, illetve szinte sosem lesz megfelelő minőségű a kapott modell az első tanítási folyamat után.

Ha így számolunk, akkor így már sokkal kisebb a felhőszolgáltatás előnye. Saját kalkulációnk szerint a saját hardver körülbelül 1 év alatt térül meg, ettől a ponttól kezdve viszont sokkal olcsóbb lesz, mint a felhő.

Folytassuk a megfelelő modell kiválasztásával.

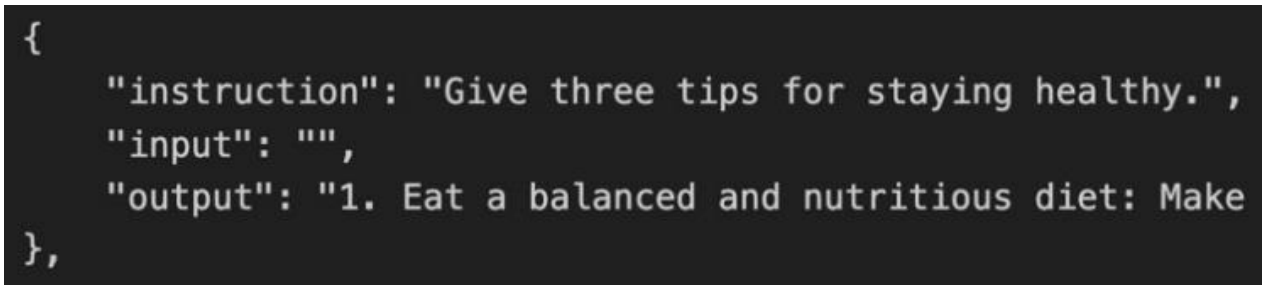
A kívánt fordítás az angol – magyar volt. Olyan modellt kellett találni, amely ezt a feladatot a lehető legjobb minőségben képes megoldani.

A Huggingface [17] oldalon van lehetőség kipróbálni is a modelleket egy webes felületen. Így könnyű eldönteni, melyik felel meg leginkább az igényeknek? Tapasztalataink szerint, erre a nyelvpárra a „Helsinki-NLP/opus-mt-tc-big-en-hu” adta a legpontosabb fordításokat, így ezt választottuk.

4. Az Adatok Előkészítése

Jelen kísérlet alapja a Huggingface-en elérhető Alpaca Cleaned adatkészlet [18]. Ez egy meglehetősen nagy szöveges adatbázis, több mint 51 000 rekorddal. Minden rekord három részből áll, ezek az instrukciók, a bemenet és a kimenet.

Hogy más nyelven is használható legyen az adatkészlet, gyakorlatilag az egészet le kell fordítani. Az instrukciók általában rövid, egy mondatos kérdések vagy utasítások, például: „Find the area of a circle given its radius.” („Adja meg egy kör területét a rádiusza alapján.”). Az input a legtöbb esetben nincs definiálva, csak akkor, ha további pontosítás szükséges. Az előbbi instrukció esetén az input ennyi: „Radius = 4”. Végül az output egy részlete: „The formula to find the area of a circle is...”. Ez a három fő összetevője van az egyes tanító adatoknak.



1. ábra. Példa adatok az adatkészletből.

Mielőtt a fordító modellt futtatni lehetett volna az adatkészleten, szükség volt előfeldolgozásra. Az instrukciók a legtöbb esetben nem voltak túl hosszúak, viszont az output sokszor elérte akár az 1000 karakteres vagy ennél nagyobb hosszúságot is. Ilyen hosszú inputokat a kiválasztott fordító modell nem tudott feldolgozni.

Habár a bemeneti adatok túl hosszúak voltak a modell számára, azokat nem lehetett csonkolni, mivel ez adatvesztéssel és a mondatok értelmetlenné válásával járt volna. A megoldás egy olyan speciális előfeldolgozó függvény készítése volt, amely információ-vesztés nélkül képes kisebb részekre osztani a szöveges tartalmat.

Ez a függvény (neve preprocessor) először megvizsgálja, hogy a kapott szöveg hosszabb-e a maximálisan megengedettnél? Ha nem, akkor egy listában visszaad egy könyvtárat, amelyben “input” kulcs alatt szerepel a teljes szöveg. Ha viszont hosszabb a szöveg, akkor először mondatokra bontja, majd egy pufferbe tölti a mondatokat, egészen addig, amíg nem lesz hosszabb a mondatok összesített hossza a megengedettnél. Ha eléri ezt a hosszt, akkor összesíti a puffer tartalmát, majd új elemként hozzáadja a kimeneti listához. Végül visszaadja a teljes listát.

```python
def preprocessor(raw_text = '', size=1000):
    if len(raw_text) <= size:
        return [{'input': raw_text}]

    parts = []
    buffer = []
    output_length = 0
    sentences = sentence_tokenizer.tokenize(raw_text.strip())
    for i in range(0, len(sentences)):
        if output_length > size:
            parts.append({'input': ''.join(buffer)})
            output_length = 0
            buffer = []
        buffer.append(sentences[i])
        output_length += len(sentences[i])

    if len(buffer) > 0:
        parts.append({'input': ' '.join(buffer)} )

    return parts
```

2. ábra. Az előfeldolgozó függvény

Ezzel a megoldással sikerült elérni, hogy a fordítónak ne kelljen túl hosszú szövegekkel dolgoznia, viszont így sem lett minden bemenet egyformán hosszú. Ez azért van, mert a mondatokat nem lehet megszakítani, ugyanis akkor a fordítás értelmetlen lenne.

Ebből is látszik, hogy az adatkészletek normalizálása és előfeldolgozása nem standard folyamat, mindig függ attól, hogy milyen környezetben és modellel kell azokat feldolgozni.

5. A Fordító Futtatása

A saját implementáció elkészítésére a Python nyelv használata volt a legkézenfekvőbb.

Ahhoz, hogy ki lehessen használni a gép képességeit, fontos volt a megfelelő operációs rendszer kiválasztása és a szoftveres környezet beállítása.

Az operációs rendszer az Ubuntu 22 volt, mivel igen stabil és a szükséges szoftverek elérhetőek hozzá.

Ide az alapvető szoftvereket is fel kellett telepíteni. A kódok szerkesztéséhez a Visual Studio Code-ot használtuk. Ezenkívül a legújabb Python verziót, az Nvidia drivereket és az Nvidia Cuda fejlesztői készletet is telepítettük.

Az elkészült alkalmazás részletesen paraméterezhető, így könnyebb megtalálni az optimális beállításokat. A főbb futtatási argumentumok a következők:

• file: a forrásfájl, amely az adatkészletet tartalmazza

• device: az eszköz, amin a fordítás fut. Három választási lehetőség van: cpu, cuda, mps. Az mps az Apple M1 és M2 eszközök esetén a beépített videókártyán való futtatást jelenti.

• cudacore: cuda eszköz esetén a futtató videókártya indexe. A cuda lehetővé teszi a videókártya kiválasztását, ha több kártya is van a gépben. Ezeket nullától kezdődően indexeli. Ha például a második videókártyán kell futnia a fordításnak, akkor a „cuda:0” paramétert kell átadni amikor az eszközt inicializáljuk.

• start: az adatkészlet kezdő indexe. Mivel a fordítási folyamat több órát vett igénybe, így lehetővé tettük az adatkészlet darabolását.

• num: az egyszerre feldolgozott adatkészlet mérete, vagy hossza.

• chunk_size: az egyes bemeneti string-ek maximális hossza.

• batch_size: a fordítást végző pipeline által párhuzamosan feldolgozott fordítási utasítások száma. Nagyobb batch_size növeli a feldolgozó processzor terhelését és a pillanatnyi memóriaigényt is, viszont gyorsítja a folyamatot.

```
# Parse arguments and set default values
parser = argparse.ArgumentParser()
parser.add_argument(
    '-f',
    '--file',
    type=str,
    help="enter json file path",
    nargs='?',
    default='',
    required=True
)
```

3. ábra. A parancssori argumentumok értelmezése

A kísérlet végén az alábbi beállítások bizonyultak a leginkább optimálisnak az adott hardveren.

A fordításért a „run_translate” függvény felelt. Ez került meghívásra a szükséges paraméterekkel. Itt a „tqdm” csomagot használtuk a folyamat követéséhez. A fordítás megkezdésekor a „start” és „stop” paraméterek alapján elvágtuk a teljes adatkészletet, majd egy flattening eljárással, hogy az összes fordítandó adat egy egydimenziós listába kerüljön. A fordítás során minden lépésnél eltároltuk, hogy a „preprocessor” hány darabra osztotta szét a bemeneti szöveget. Majd az egyes string-ek fordítása után a segédváltozó segítségével állítottuk vissza az eredeti struktúrát.

A tesztelést CPU-n és Apple Silicon M1 rendszeren is elvégeztük. A teljesítmények összehasonlítása az alábbi táblázatban látható.

1. Táblázat. Teljesítmény összehasonlítása

Eszköz	*iteráció / s*	*fogyasztás (W)*	*W / it*
Cuda	20	600	30
CPU	1.5	250	167
Apple Silicon M1	1.5	8	5

Meglepődve tapasztaltuk, hogy egy Apple gyártmányú laptop milyen hatékony tud lenni a beépített ARM architektúrájú processzorral. Ugyan sokkal lassabban végezte el a feladatot, viszont a ráfordított elektromos munka csupán a hatoda volt, mint a Cuda architektúra esetén.

6. Következtetések

A fő kérdés az, hogy megéri-e saját hardveren futtatni az ilyen tipikus fordítási feladatokat? A válasz az, hogy igen, amennyiben nagy mennyiségű szöveget szeretnénk lefordítani. Összehasonlítva a saját megoldást a Deepl vagy Google Translate platformokkal, lényegesen olcsóbb a saját hardver használata. Például a Deepl API használata esetén jelenleg 1 000 000

karakter fordítása 20 Euro-ba kerül. A vizsgált adatszett önmagában 50 000 000 karakter, így ez 1000 Euro lett volna. Azaz a saját hardver ára két adatszett fordítása során megtérül.

De mi az eredmény, ha a szkriptet a felhőben futtatják? Így már közel egy évre tolódik a saját hardver megtérülésének idelye. Viszont a felhőszolgáltatás hátránya a speciális platformok használata, amivel a szolgáltaók magukhoz láncolják az ügyfeleket.

Mennyire lett jó minőségű a fordítás? Ezt talán a legnehezebb mérni, így empirikus vizsgálattal, szúrópróbával mértük a pontosságot. A minőség a Google Translate és a Deepl közé esett, a legtöbb esetben elfogadható volt. Itt meg kell jegyezni, hogy nem biztos, hogy megtaláltuk a tökéletes modellt, egy másik modell alkalmazásával tovább javulhat a fordítás minősége.

A célkitűzésünket elértük, saját adatszettet fordítottunk magunknak saját hardveren. Így olyan minta alkalmazás készült, amelynek a segítségével a későbbiekben más ingyenes adatkészleteket is le tudunk fordítani.

Alátámasztottuk, hogy nagy mennyiségű szöveg fordítása esetén megtérül a saját hardverre való beruházás.

Ezenkívül mérésekkel igazoltuk, hogy a jelenleg leginkább hatékony MI platform az Apple által fejlesztett ARM alapú chip, habár a teljesítménye jelenleg még elmarad az Nvidia Cuda alapú megoldásoktól, de a fogyasztása csak mintegy 17%-a annak, azonos hosszúságú szöveg fordítása esetén.

Referenciák

[1] Youssoufa Mohamadou, Aminou Halidou & Pascalin Tiam Kapen, A review of mathematical modeling, artificial intelligence and datasets used in the study, prediction and management of COVID-19, Applied Intelligence Volume 50, 2020, pp. 3913–3925, DOI: 10.1007/s10489-020-01770-9

[2] Shibin David, R.S. Anand, Martin Sagayam., Enhancing AI based evaluation for smart cultivation and crop testing using agro-datasets, Journal of Artificial Intelligence and Systems, 2, 2020, pp. 149–167, DOI: 10.33969/AIS.2020.21010

[3] György, Molnár ; Cserkó, József. AI Based Plagiarism Checking, In: Molnár, György (ed.) IEEE 5th International Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE 2022), IEEE 2022, pp. 000187-000191.

[4] Nemeskey Dávid Márk, An exercise to test your mettle. In: MSZNY 2020, XVI. Hungarian Conference on Computational Linguistics. Szeged: University of Szeged, Department of Informatics, pp. 409-418.

[5] Molnar G., Szuts Z., Use of Artificial Intelligence in Electronic Learning Environments In: Molnár, György (ed.) IEEE 5th International Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE 2022) Budapest, Magyarország : IEEE (2022) pp. 137-140. , 4 p.

[6] Szűts, Zoltán; Geoffrey, Vaughan. The Educational Challenges of ChatGPT: előadás = Lecture (2023) 12th International Conference on Applied Informatics (ICAI 2023) 2023-03-02

[7] Balogh, Zoltán; Fodor, Kristián ; Martin, Magdin ; Jan, Francisti ; Štefan, Koprda ; Kővári, Attila. Development of Available IoT Data Collection Devices to Evaluate Basic Emotions, ACTA POLYTECHNICA HUNGARICA 19 : 11, 2022, pp. 165-184.

[8] György, Molnár; Cserkó, József; Karl, Éva. Evaluation and technological solutions for a dynamic, unified cloud programming development environment: Ease of use and applicable system for uniformized practices and assessments In: Szakál, Anikó (szerk.) IEEE 21st World Symposium on Applied Machine Intelligence and Informatics SAMI (2023) : Proceedings Herlany, Szlovákia : IEEE (2023) pp. 237-240.

[9] Kulkarni, A., & Shivananda. A.. Natural Language Processing Recipes: Unlocking Text Data with Machine Learning and Deep Learning using Python. Apres, 2019, DOI: 10.1007/978-1-4842-4267-4

[10] Maximilian Hopf. “NLP With Google Cloud Natural Language API”, https://www.toptal.com/machine-learning/google-nlp-tutorial Last download: 18.04.2023

[11] Christopher Pappas. “Why eLearning Pros Should Use Predictive Analytics”, https://elearningindustry.com/ Last download: 18.04.2023

[12] Maftuna Khidirova. “Digitalization of testing and assessment”, AoC, vol. 17, no. 1, 2021, pp. 226-229

[13] Szabó, Csilla Marianna, Bartal, Orsolya, Nagy, Bálint. The Methods and IT-Tools Used in Higher Education Assessed in the Characteristics and Attitude of Gen Z, 2021, DOI: 10.12700/APH.18.1.2021.1.8

[14] Beáta Orosz. Learner Experiences Related to Digital Education Schedules in Light of Empirical Data, 2021 ACTA POLYTECHNICA HUNGARICA.18.1.2021.1.9

[15] Mónika, Garai-Fodor, The Impact of the Coronavirus on Competence, from a Generation-Specific Perspective, ACTA POLYTECHNICA HUNGARICA 19 : 8 pp. 2022, pp. 111-125.

[16] Amazon SageMaker Pricing, Last download: 23.08.2023, https://aws.amazon.com/sagemaker/pricing/

[17] HuggingFace, Last download: 23.08.2023, https://huggingface.co/

[18] HuggingFace – Alpaca Cleaned Dataset, Last download: 23.08.2023, https://huggingface.co/datasets/yahma/alpaca-cleaned