Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Általános Informatikai Intézeti Tanszék

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék

T5 alapú kérdésgeneráló mintarendszer fejlesztése és hatékonyságelemzése

Diplomamunka

Készítette:

Név: Megyeri Balázs

Neptunkód: AXQBOZ

Szak: Mérnökinformatikus MSc
Alkalmazás fejlesztő szakirány

Eredetiségi Nyilatkozat

Alulírott **Megyeri Balázs**; Neptun-kód: `AXQBOZ` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Mérnökinformatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Automatikus kérdés generálás magyar nyelvű szövegekből* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....

Hallgató

Tartalomjegyzék

1. fejezet

Bevezetés

A természetes nyelvfeldolgozás az informatika egyik talán legkomplexebb feladatköre. Ennek egyik oka, hogy az emberi nyelv és annak kialakulása szorosan összefügg az emberi aggyal és annak evolúciójával, melyet még a mai napig se sikerült teljesen feltérképeznünk és megértenünk. Nyelvünk értő használata egyike azon utolsó problémaköröknek, amiket a számítógépek eddig nem voltak képesek még megközelítőleg se megfelelően teljesíteni, hiszen akár már egy egyszerű mondat feldolgozásához, kontextusban való elhelyezéséhez vagy akár kibővítéséhez is óriási méretű szabályhalmazokra és számítási teljesítményre van szükség.

Az utóbbi időkben azonban jelentős sikereket tudtak elérni a gépek más területeken. Egyre elterjedtebbé váltak a különböző objektumfelismerő algoritmusok, melyek akár alacsony minőségű képekből is képesek felismerni alakzatokat, formákat és mindezt közel valós időben mozgás közben is. Ezen algoritmusoknak már számos vállalati és állami felhasználása is van.

Továbbá megjelentek különféle képgeneráló algoritmusok, melyek generatív versengő hálózatok(GAN hálózat) segítségével művészi minőségű képeket tudnak generálni. Számos weboldal készült már, ahol pár sor szöveg megadása után a szerver generál egy képet a megadott szöveg alapján és megjeleníti azt tetszőleges minőségben. Ez a megoldás már szövegfelismerést is tartalmaz, valamint a szöveg egyes részeihez csatolt képi alakzatokat, melyek segítségével mondjuk egy GAN hálózat megkonstruálhatja a kívánt képet, akár csak egy igazi művész.

De vajon képesek-e a gépek magát a szöveget elkészíteni egy ilyen képgenerálóhoz? Képesek-e írói minőségű vagy kulturális igényű szövegeket készíteni? Tudnak-e kérdéseket megfogalmazni vagy éppen válaszokat? Ezekre a kérdésekre fogom keresni a választ diplomamunkámban.

Továbbá egy konkrét webalkalmazás fejlesztésén keresztül bemutatom a jelenleg rendelkezésünkre álló fejlesztői környezetet és könyvtárakat, valamint a szöveggenerá-

lás során jelenleg használt legfejlettebb algoritmusokat is. Az alkalmazás képes lesz egy adott kontextus alapján nyelvtanilag helyes és releváns kérdéseket generálni, melyek utána szabadon felhasználhatóak lesznek. Célunk a programmal, hogy elősegítsük oktatási segédletek, tananyagok, feladatok készítését. A diákok számára is nagy segítséget nyújthat az alkalmazás a tanulás során, hiszen egy-egy szövegrészlet alapján generált kérdésekkel remekül összefoglalhatóvá válik azok tartalma, mondanivalója. Mindezek mellett a szoftver felhasználható vállalati tevékenységek során felvételi interjúkban, tesztek során, workshop-okban vagy egyéb tervezést vagy mélyebb megértést igénylő feladatok során is, de működése jó alapot szolgáltathat akár egy chatbot kérdés vagy témageneráló komponensének elkészítéséhez is.

2. fejezet

Természetes nyelvfeldolgozás

2.1. Történeti áttekintés

Az emberi nyelv egy komplex médium gondolatok, információk, ötletek és érzelmek átadására, továbbítására. Nagyon nehéz ezt a működést matematikai formulákkal, képletekkel leírni. A legegyszerűbb mondatok leírása is több oldalas feladat lehet formális nyelveket használva. Emiatt különösen nehéz dolguk van a gépeknek az emberi nyelvek értelmezésével, vagyis a természetes nyelvfeldolgozással(NLP - Natural Language Processing).

Az "fordító gép" fogalom első előfordulása az 1930-as évek közepére tehető. Akkoriban két szabadalom is létezett a technológiára. Az első szabadalom **Georges Artsrouni** nevéhez köthető, aki egy kétnyelvű szótárat használt arra, hogy átfordítsa a szavakat közvetlenül egyik nyelvről a másikra egy papírszalag segítségével. Ez egy nagyon kezdetleges megoldás volt, mivel a nyelvtani különbségekkel nem tudott mit kezdeni. A második egy orosz szabadalom volt **Peter Troyanskii** nevéhez fűződően. Ő szintén egy kétnyelvű szótár felhasználásával próbált fordítani, azonban ő figyelembe vette az egyes nyelvtani szabályokat is. Mindkét megközelítés hasznosnak bizonyult technikai szempontból, azonban működő modellt nem igazán sikerült készíteniük, inkább koncepcionális megoldások voltak.[1]

Az első kísérlet az NLP alkalmazására a németekhez köthető a 2. világháború alatt. Ők fejlesztették ki az **Enigma** nevű gépezetet, melyet titkos üzenetek kódolására használtak. A gép képes volt kódolni, illetve továbbítani az egyes parancsnokoknak és katonai egységeknek szánt üzeneteket. Később erre válaszul az angolok elkészítették a **Colossus** nevű gépet, amely képes volt dekódolni az Enigma által kódolt üzeneteket, így járulva hozzá a szövetségesek későbbi győzelméhez. A második világháború alatt az angolok kriptográfiai kutatásai elsősorban a Bletchley Park-ban zajlódtak. Itt dolgozott Alan Turing is kollégáival, akihez később számos új megközelítés is kötődik az

informatika történetében.[1]

1950-ben **Alan Turing** megalkotta a Turing-tesztet, ami úttörővé vált a természetes nyelvfeldolgozás területén. A teszt lényege, hogy eldöntse egy gépről tud-e emberhez hasonlóan gondolkodni. Magához a teszthez 3 személyre van szükség: 1 férfira, 1 nőre és 1 kérdezőre. A kérdezőt elszeparálják a játékosoktól. A teszt során a kérdező megpróbálja meghatározni a másik két személy nemét kérdések és rájuk adott válaszok által írásban. A csavar a tesztben, hogy az egyik személy a helyes megoldás felé próbálja terelni a kérdezőt, míg a másik próbálja átverni őt és a helytelen megoldás felé vezetni. Turing azt javasolta, hogy ezt a játékost cseréljék le egy gépre. Ha a kérdező sikeresen meg tudja határozni mindkét játékos nemét, akkor a gép elbukott a Turing-teszten, egyébként pedig átment rajta. Maga a teszt nem szimplán arról szól, hogy a gép meg tudja-e oldani ezt a problémát, hanem hogy eldöntse tud-e olyan feladatokat végezni a gép, amit csak egy ember tud, vagyis hogy képes-e emberként gondolkodni.

Ahhoz, hogy a gépek képesek legyenek megérteni az emberi nyelveket elengedhetetlen a megfelelő nyelvtanok alkalmazása. Az egyes mondatok értelmezéséhez a gépnek ismernie kell a különböző nyelvtani szabályokat, vagyis tudnia kell, hogy például vannake ragok az adott nyelvben, milyen igék, tárgyak vannak, illetve ismernie kell a különböző mondathatároló karakterek jelentéseit. 1957-ben **Noam Chomsky** könyvében[3] bevezette a szintaktikai szerkezetek fogalmát. Munkájában nagy hangsúlyt fektetett a nyelvi szerkezetek formalizálására. A természetes nyelveket is el tudta helyezni egy hierarchiában, melynek köszönhetően elkezdődhetett az NLP feladatok gépeken történő megvalósítása. A későbbiekben Charles Hockett számos hátrányt fedezett fel Chomsky megközelítésében, mivel az egy jól meghatározott és stabil struktúrát és formális rendszert tételezett fel a nyelvek mögött, ami az emberi nyelvekre csak kivételes esetekben volt igaz.

Az NLP-t legelőször a gépi fordításban használták. A gépi fordítás lényege, hogy olyan programokat készítsünk, melyek képesek egyik emberi nyelven írt szövegről egy másik emberi nyelven írt szövegre fordítani, akár valós időben is. Ilyen fordító volt 1954-ben a **Georgetowni Egyetem** és az **IBM** által közösen fejlesztett program[2] is, ami 60 orosz nyelvű mondatot is képes volt angolra fordítani. Működése egyszerű volt: szótár használatával közvetlenül fordította a mondatokat egyik nyelvről a másikra. Ezt a szótárat pedig a program készítői felügyelték és tartották karban. A készítők nagy elvárásokat támasztottak programjuk felé, azonban financiális okok miatt végül abba kellett hagyniuk a projektet.[1]

1960-ban **Terry Winograd** elkészítette **SHRDLU** nevű programját[1], ami egyike volt az első NLP-t használó programoknak. A programnak lehetett különböző utasításokat adni, hogy nevezzen meg objektumokat egy képen, mozgasson alakzatokat, illetve

le lehetett benne kérdezni az aktuális állapotot a blokkokból álló virtuális világában. A szoftver lenyűgözte a mesterséges intelligenciával foglalkozó szakembereket és számos új megoldást inspirált, azonban komplexebb, valós világból származó problémák megoldására nem igazán lehetett használni.

1969-ben **Roger Schank** bevezette a tokenek használatát a természetes nyelvfeldolgozásban.[1] Az egyes tokenek különböző valós világbeli objektumokat, cselekvéseket, helyeket és időt jelöltek. Ezen tokenek segítségével a gép könnyebben megtudta érteni az egyes mondatok jelentéseit. Ez a tokenes megoldás a mai napig használatban van és részben példaprogramunkban is használni fogjuk.

Az eddigi felvázolt megoldások mindegyike nyelvtani szabályok és struktúrák alapján próbálta értelmeztetni a géppel a mondatokat, azonban tudjuk, hogy pusztán ezek ismerete nem elég egy adott mondat helyes feldolgozásához. Pontosan emiatt 1970-ben **William Woods** bevezette az ún. kiterjesztett átmeneti hálózatokat(**ATN**) a természetes nyelvek reprezentációja során.[4] Működésének lényege, hogy az elérhető információk felhasználásával véges automatákat használt rekurzióval a mondatok értelmezéséhez. Tehát a program ad egy lehetséges megoldást az adott szöveg jelentésére és ahogy egyre több információt adunk meg úgy kezdi el javítani, finomhangolni a jelentést is. Amíg nem biztosítunk elég információt a hálózat számára, addig rekurzióval próbál megoldást találni vagy képtelenné válik biztos jelentés meghatározására. Ez a rekurzió szerű működés megfigyelhető napjaink szöveggeneráló és chatbot alkalmazásaiban is.

A közelmúltban új trendek kezdtek el megjelenni az NLP területén. A korábbi szigorú kézi szabályhalmazokat alkalmazó megoldásokat elkezdték háttérbe szorítani a különböző **gépi tanulást** használó valószínűségeken alapuló algoritmusok, melyek első jelentősebb felfutása az 1980-as évekre tehető. Ilyen algoritmusok voltak például a döntési fák, melyek ha-akkor szabályok alkalmazásával képesek voltak optimalizálni az egyes NLP feladatok eredményeit.

Napjainkban a figyelem elsősorban a **mély tanulást** alkalmazó megoldások felé irányult, ami nem is lehet véletlen, hiszen ezek a megoldások a neurális hálózatok használatával az ember információfeldolgozó képességét próbálják lemásolni és gépekre átültetni. Ezen megoldások lényege, hogy ne próbáljunk meg fix szabályokat vagy formulákat megadni a gépnek egy szöveg értelmezésénél, hanem mutassunk példákat a különböző nyelvi elemekre és alakítsa ki a gép magának ezeket a szabályokat és összefüggéseket. Mindezen változtatásokra a probléma megközelítésében azért volt szükség, mert a természetes nyelvfeldolgozás során számos olyan nehézséggel találkozhatunk, melyek más formálisabb, kötöttebb területeken egyszerűen nem jelennek meg. Ezen problémaköröket fogom ismertetni a következő alfejezetben.

2.2. A természetes nyelvfeldolgozás nehézségei

Mint minden szakterületnek, így a természetes nyelvfeldolgozásnak is vannak nehézségei vagy akár adott technológiával pillanatnyilag megoldhatatlan feladatai. Maguknak a természetes nyelveknek a gépek általi megértése is ilyen megoldhatatlannak gondolt probléma volt a 20. században, hiszen talán ez az az utolsó válaszvonal az emberek és a gépek között, melyet átlépve már a technológiai szingularitás küszöbére kerül az emberiség. Továbbá ez az a szakasz, ahol teljesen egyértelműen meg tudunk különböztetni egy emberi és egy gépi agyat. De melyek is azok az egyes problémakörök, melyek alapján jogosan gondolhatnánk lehetetlennek a gépek számára az emberi nyelvek megértését?

Az első ilyen nehézség az a **többértelműség**. Bizonyos szavak szándékos vagy nem szándékos módon többféle jelentéssel is bírhatnak számunkra. Ez adódhat abból, hogy egy adott szó átvételre került egy másik nyelvből és ütközik egy már meglévő, de más szófajú szóval. Ilyen például a "vár" szavunk, amit használhatunk igeként és főnévként is. Ebben az esetben nem szándékos többértelműségről beszélünk. De akadhat olyan eset is például a szépirodalomban, ahol igenis direkt módon van használva a többértelműség. Ilyen alkalmazását találhatjuk meg például Kosztolányi Dezső *Aranysárkány* című művében, ahol a mű központi alakja Novák Antal vitatkozik, hogy mit jelent a diákok által készített magasban repülő sárkány. Novák játéknak gondolja, míg Fóris fenyegető hatásúnak. Nézetkülönbségük a "sárkány" szó kétértelműségén alapul, ami jelenthet reptetésre való papírsárkányt és ősi mítoszokból eredő tűzokádó teremtményt is. Ez a példa egyben a műfordítás egyik problémáját is felveti, hiszen, ha ezt a szöveget angol nyelvre szeretnénk átfordítani, akkor bajban lennénk, hiszen az angol nyelvben külön szó létezik a papírsárkányra(kite) és az állati sárkányra(dragon), így nem lenne értelmezhető a két szereplő vitája.

Láthatjuk, hogy számos nehézség következik a kétértelműségből és így, ha NLP-vel foglalkozunk, akkor kezdenünk is kell vele valamit. De hogyan tudnánk megoldani, hogy a gép el tudja kerülni ezt a problémát és helyesen értelmezzen szépirodalmi szövegeket? Vegyük példának ezt a mondatot:

„A szolgáltatónak kell fizetni."

Ez a mondat 2 különböző jelentést is takarhat:

- Valakinek be kell fizetnie egy bizonyos díjat egy szolgáltatónak.
- Magának a szolgáltatónak kell kifizetnie egy adott összeget valakinek.

Mind a 2 értelmezés helyes szintaktikailag, azonban szemantikailag nem mindegy, hogy hogyan értelmezzük. Természetesen a mondat pontos jelentése egyértelművé válik,

amint megismerjük a kontextust melyben a mondat elhangzott, de mindehhez komplex háttértudásra van szükségünk. Ennek a háttértudásnak az ismerete hiányzott eddig a különböző NLP feladatok megoldására írt programokból, hiszen ezek rengeteg adatot, metaadatot, szabályt és egyéb heurisztikát igényelnek. Mi emberek az evolúció, illetve az egyéni fejlődés során gyerekkortól megismertük ezt a szükséges háttértudást egy ilyen mondat értelmezéséhez, viszont a gépek nem rendelkeztek eddig az ezekhez szükséges eszköztárakkal. Tehát a megoldás, hogy valamilyen módon példákat kell mutatnunk a gépnek ezekre az esetekre és tanítanunk kell folyamatosan, hogy el tudja dönteni a kontextus alapján ezen szövegek jelentését.

Egy további nehézség lehet a természetes nyelvfelismerésben az **apró részletek** és a **szórend** okozta különbségek az értelmezésben. Sokszor egyetlen szó, de akár egy betű vagy írásjel is teljesen megváltoztathatja egy mondat jelentését. Tekintsük mondjuk ezeket a példákat:

„Lőttem egy gyönyörű fotót."
„Lőttem egy gyönyörű vadat."

Amellett, hogy a "lőttem" szó többértelmű és ez önmagában is okozhat problémákat vegyük észre, hogy a két mondat csupán egyetlen szóban különbözik. Ebben az esetben, ha például egy korábbi megoldással egy koszinusz hasonlósági számítással szeretnénk értelmeztetni a géppel ezt a mondatot és el szeretnénk helyezni a mondatok egy bizonyos csoportjában akkor ez a két mondat jelentését tekintve nagyon közel kerülne egymáshoz. Tehát a gép számára bizonyos hibahatárok között ugyanazt jelentené a két mondat, annak ellenére, hogy két teljesen más jelentésről van szó. Mindezek miatt szükségessé vált, hogy a gépet folyamatosan tanítsuk újabb példákkal, hiszen maga a nyelv is folyamatosan fejlődik. Korábban a "lőttem" szó tényleges lövést jelentett, ma pedig már egy fotó elkészítését is jelentheti. Tehát egy olyan mechanizmusra van szükségünk, amit nem elég egyszer elkészítenünk vagy betanítanunk, hanem rendszeresen frissíteni kell a tudását az idők során.

Újabb problémákat vetnek fel a szépirodalomban megtalálható **költői képek**, mint a metafora, az allegória, a metonímia vagy a különböző szimbólumok értelmezése. Ezek értő használata még az emberek között is a legmagasabb kulturális szintnek felel meg, így ezeket a gép se fogja egyszerűen megérteni és használni. Ez a problémakör ráadásul nem csak a természetes nyelvfelismerést érinti, hanem például a képfeldolgozást is. Ugyanis ezek a művészeti eszközök megjelenhetnek a szobrászatban vagy a festészetben is. Számos példa volt már a gyakorlati felhasználásában ezeknek a képfelismerő algoritmusoknak, ahol mondjuk meztelenséget kellett volna az algoritmusnak kiszűrnie egy adott képen, azonban olyan képeket is szimplán meztelenségnek kezdett el érzé-

kelni, ahol egy szobor vagy egy festmény, egy művészeti alkotás volt látható. Itt a gép láthatóan nem volt képes a meztelenségnek, mint alkotói eszköznek, a szabadság, az újjászületés vagy a tisztaság szimbólumának a megértésére. Ugyanez igaz a szövegfeldolgozásra is, ahol például egy szimpla szó, mint a "tenger" Petőfi Sándor *Föltámadott a tenger* című versében egyszerre jelenti a valódi nagy kiterjedésű víztömeget, illetve a népek tömegét. Viszont a gép nem tudja jelenleg ezt a komplex kapcsolatot feltárni a nép és a tenger között akármennyi példát is mutatunk rá neki. Ez a kapcsolat akkor is egy hosszú megértési folyamatnak lesz az eredménye, mely magába foglal történelmi, művészeti, nyelvi és érzelmi tudást.

Az **irónia** és a **szarkazmus** is számos félreértés tárgya lehet a különböző NLP algoritmusoknak, de akár még egyes emberi moderátoroknak is. A két fogalmat gyakran keverik, illetve mossák össze, és bár valóban van közös metszetük, de alapvetően eltér a jelentésük. Az irónia azt jelenti, hogy a szöveg szó szerinti értelme és a tényleges, a beszélő szándéka szerinti értelme ellentétes. Itt már rögtön találkozunk egy NLP szempontjából elsőre nehezen értelmezhető fogalommal a "szó szerinti" jelentéssel. Ez a probléma a gép számára jelentős kihívást jelent, hiszen ha adunk a gépnek egy mondatot és megkérdezzük tőle a jelentését, akkor a gép elsőre helyesnek tűnően fogja megválaszolni nekünk a jelentést, azonban mi tudni fogjuk, hogy ez a jelentés nem pontos. A szarkazmus ezzel szemben csak annyit jelent, hogy a beszélő szándéka egy adott kifejezéssel, hogy gúnyoljon valakit vagy valamit vicctől és humortól mentesen, de mégis a sorok között elbújtatott formában. Tehát a gépnek nem elég egyetlen jelentést számon tartania az egyes mondatokról és kifejezésekről, hanem tudnia kell az összes lehetséges jelentését az adott szövegnek. Vegyük például a következő mondatot:

„Na jól állunk!”

Ennek első jelentése, hogy jól haladnak a dolgok, tehát valami pozitív történt vagy történik. Második, valódi jelentése azonban pont ellentétes, tehát rosszul állnak a dolgok, negatív a kontextus. Ez az ellentét a gép szempontjából értelmezhetetlennek tűnik, azonban itt is, akár csak a többi nehézség esetében a kontextusok megtanulása megoldhatja ezen problémákat.

Utolsó fontosabb problémakörünk a különböző **szöveghibák**, illetve az egyes nyelveket érintő **forráshiány**. Az írott, illetve diktált szövegekben gyakoriak az elírások és a helytelenül használt szavak. Ezek egyértelműen megváltoztatják vagy egyenesen értelmezhetetlenné teszik a szövegek feldolgozását. Ezen hibák oka lehet a figyelmetlenség, az akcentus vagy az esetleges dadogás. Ilyenkor használhatunk különböző nyelvtani javítóprogramokat a bemeneti szövegeken, azonban itt se garantált a tökéletes működés. Ez a probléma is rámutat arra, hogy egy ilyen NLP feladatot megoldó programnak

több különböző problémacsoportot kell tudnia kezelni és nem elég szimplán szövegeket megtanulnia.

A másik probléma, ami engem is érintett diplomamunkám gyakorlati része során az a szöveges forráshiány bizonyos nyelveken. Ahhoz, hogy az NLP feladatokat megvalósító modern alkalmazásaink megfelelően működjenek elengedhetetlen, hogy az adott nyelveken jól szűrt és kedvezően formázott szöveges forrásaink vagy bemeneteink legyenek. Ez azért fontos, mert később a program a tudásbázisát ez alapján a bemenet alapján fogja felépíteni és a fentebb felsorolt problémákat is ez alapján kell majd neki felismernie és megoldania. Például ahhoz, hogy angol nyelven tudjunk kérdéseket generáltatni egy programmal, ahhoz szükség lehet egy olyan angol nyelvű szöveges forrásra, ahol adott egy témakör és egy szöveges kontextus, valamint adottak hozzá illő kérdések. Ha ez nem áll rendelkezésünkre, akkor máris egy hatalmas problémával találjuk szembe magunkat, hiszen nem fogunk tudni alkalmas példákat mutatni a programunknak az adott feladat megvalósításához, illetve a tesztelést is meg fogja nehezíteni.

2.3. NLP feladatok

Mivel az emberi agy digitális újraalkotása egy meglehetősen nehéz és bonyolult feladat, így a különböző csak emberek által elvégezhetőnek vélt NLP feladatokat külön-külön csoportokba és alcsoportokba szokták bontani. Természetesen a jövőben elkészülhet egy olyan mesterséges intelligencia, ami már egyben képes lesz az összes NLP feladat elvégzésére, de egyelőre itt még nem, vagy csak részben tartunk.

Az első fontosabb NLP terület az a **szintaktikai elemzés**. Ennek során az algoritmusnak azonosítania kell az adott szöveg szintaktikai struktúráját és fel kell tárnia az egyes szavak és mondatok függőségi relációját. Ezen folyamat eredménye lehet például egy ún. elemzési fa, mely egy rendezett és gyökérelemmel rendelkező fa-struktúra, amelyről leolvasható lesz az adott szöveg szintaktikai struktúrája valamilyen kontextusfüggetlen nyelvtan szerint. Ezen feladat eredményeit fel lehet használni az információkinyerés, gépi fordítás, illetve a nyelvtani ellenőrzés, javítás területén.

Egy másik NLP terület a **szemantikai elemzés**, mely már az adott szöveg tényleges jelentésére fókuszál. A fentebb felsorolt NLP problémák miatt talán ez a feladat tekinthető a legnehezebbnek. A szemantikai elemzéshez kapcsolódó feladatok során elemezzük a mondatok struktúráját, az egyes szavak között lévő interakciókat és a kapcsolódó fogalmakat, annak érdekében, hogy feltárjuk a szavak jelentését, illetve az egész szöveg témáját és kontextusát.

A legtöbb NLP feladatban szükségünk van arra, hogy a megadott mondatainkat és szavainkat a gép számára is értelmezhető formátumra alakítsuk. Ekkor lehet segítsé-

günkre a **tokenizálás**. A tokenizálás egy alapvető feladat az NLP-ben, melynek során egy adott szöveget szemantikailag hasznos egységekre bontunk fel, melyeket később fel tudunk használni algoritmusainkban. Lényegében lefordítjuk a szavakat és mondatokat egy optimális, a gép által is értelmezhető nyelvre. A művelet során viszonylag nagy szabadságunk van a formátumot illetően, de általában a szó tokeneket szóközökkel, a mondat tokeneket pedig valamilyen egyedi karakterrel vagy karaktersorozattal szokták jelölni.

NLP feladatok megoldása során szükségünk lehet arra, hogy a szöveg egyes egységeit megcímkézzük egy adott kategóriával, ezzel segítve a gép számára a megértést. Erre szolgál a **beszédrész-címkézés**(Part-of-speech tagging). Ennek során a szavakhoz vagy akár az előző feladat alapján generált tokenekhez is hozzárendelhetünk különböző kategóriákat, mégpedig az alapján, hogy az adott egység milyen szerepet tölt be a mondatban. A legelterjedtebb kategóriák közé tartoznak az igék, főnevek, melléknevek, névmások és kötőszavak, de saját egyedi kategóriákkal is elláthatjuk az egységeket.

Bizonyos nyelvek, mint például az angol vagy a magyar tartalmaznak olyan nyelvi elemeket, melyek egy adott feladat megoldása során a gép számára szükségtelenné válhatnak. Ilyen elemek például a ragok. Ezen szükségtelen elemek kiszűrésére és leválasztására szolgál a **lemmatizáció** és a **tőképzés**. A lemmatizáció során szavak bizonyos csoportjait(melyeknek általában azonos a szótöve) egyetlen szóban próbálunk meg leírni és a feladatok megoldása során az ezen kategóriájuk szavakhoz ezt az egy szót használjuk majd. A tőképzés folyamatának az eredménye szintén egyetlen szó lesz, azonban ott a konkrét szótő megtalálása lesz a cél és minden azonos szótövű szót erre az egyetlen szótőre cserélünk ki, így könnyítve a gép dolgát. Ezen két módszer arra a feltételezésre épül, hogy az azonos szótövű vagy hasonló kategóriába eső szavak jelentésükben is nagyon közel állnak egymáshoz és ezáltal csökkenthető a különböző szavak darabszáma a szövegben, így gyorsítva a feldolgozást.

Egy újabb feladatkör lehet a szövegfeldolgozás gyorsítására a **tiltólistás szavak** (stopwords) kiszűrése. Ennek során kiválogatjuk a szövegből azokat a gyakran előforduló szavakat, melyeknek a legkisebb a szemantikai értéke, vagyis amelyek a legkevesebbet adják hozzá a szöveg értelmezéséhez. Ezek lehetnek kötőszavak, névmások, elöljárószavak, de akár tetszőleges, az adott NLP feladat megoldásához számottevően hozzá nem tevő szavak is.

Ahhoz, hogy a gép megfelelően tudjon értelmezni szövegeket elengedhetetlen, hogy a haszontalan szövegelemek mellett a leghasznosabb részeket is ki tudja szűrni. Erre szolgál a **névelem-felismerés**(Named Entity Recognition), mely feladat során a gépnek ki kell szűrnie bizonyos szavakat vagy mondatrészeket a szövegekből és el kell helyeznie egy adott kategóriában. Ez a kategória lehet városnév, személynév, helynév,

cégnév, vagy akár email cím is, attól függően, hogy mire specializáljuk modellünket. Ezen feladat egy komplexebb formája a reláció felismerés, mely annyival bonyolítja meg az alapfeladatot, hogy nem csak egy-egy szót vagy mondatrészt vizsgál, hanem megpróbál kapcsolatokat felfedezni kettő vagy több különböző szó, illetve mondatrész között.

Az utóbbi évek talán egyik legnépszerűbb NLP feladatai közé tartozik a **szövegosztályozás**. A feladat során adottak különböző kategóriák és a gépnek el kell döntenie egy tetszőleges szövegről, hogy mely kategóriába illik. Ennek egyik legelterjedtebb és aktívan használt változata a hangulatelemzés, amikor szövegeket az alapján próbálunk meg kategorizálni, hogy pozitív vagy negatív a hangvételük, vagy hogy milyen egyéb érzelmeket tudnak kiváltani. Erre a feladatra talán a legalkalmasabb megoldások a neurális hálózatok, hiszen ezek alapvetően mintafelismerő rendszerek, így könnyen boldogulnak a feladattal.

Végezetül elérkeztünk dolgozatom fő NLP témaköréig a **szöveggeneráláshoz**. Jelenleg talán ez az NLP feladatkör az, ahol a legpopulárisabb eredményeket sikerül elérni az utóbbi időkben, hiszen a ChatGPT megjelenése széles körben elterjesztette az NLP megoldások ezen alkalmazását. Szöveggenerálásnál a gép feladata, hogy egy megadott kontextus alapján hozzon létre szöveget úgy, hogy a kontextus szerepeljen benne, de közben intelligens módon dúsítsa fel vagy éppen csökkentse a kimeneti szöveget valamilyen alfeladatnak megfelelően. A kontextus lehet egy szövegrészlet, egy szó, de akár el is hagyhatjuk és ilyenkor a gép magától alkot egy kontextust és mondjuk egy chatbot esetén beszélgetést kezdeményezhet. A szöveggenerálásnak számos alfeladata van, melyek mind eltérő megközelítést igényelnek, de alapvetően hasonló a céljuk.

Az első alfeladat a **szövegösszegzés**. Ekkor adott egy bemeneti szöveg és az algoritmusnak szimplán annyi a feladata, hogy kiszűrje belőle a feleslegesnek ítélt részeket és egy rövidebb, tömörebb szöveget adjon vissza eredményül. Ez az alfeladat alkalmas például megbeszélések, előadások rövid összefoglalójának elkészítésére, vagy akár tananyagok alapján tanulási segédanyagok generálására.

Egy újabb alfeladat a **kérdés megválaszolás**(Question answering). Ekkor a bemeneti kontextus egy kérdés, a kimenet pedig egy vagy több a kérdésre adott válasz. Ezen a területen a legtöbb megoldás természetesen valamilyen adott témakörökben képes csak kérdések megválaszolására, azonban például a ChatGPT, mivel egy majdnem az egész szöveges internetet lefedő adathalmazon lett tanítva, így képes szinte bármilyen témában kérdéseket megválaszolni, akár még formális nyelveken is, például programozási nyelveken vagy a matematika nyelvén.

Az előző alfeladathoz kötődik, de mégis teljesen más kategóriában helyezkedik el dolgozatom fő témája, a **kérdésgenerálás** feladatköre. Ennél a feladatnál adott ne-

künk egy szövegrészlet, egy kontextus, amelyhez a gépnek kérdéseket kell alkotnia. A feltehető kérdéseknek több fajtája is van melyeket az alapján csoportosíthatunk, hogy milyen típusú választ várunk rájuk[5]:

- Kiegészítendő kérdések
- Eldöntendő kérdések
- Választó kérdések

Ideális esetben az algoritmus mind a 3 kategóriában képes lesz kérdéseket generálni, de itt is, akár csak a többi esetben a tanítóhalmaz mérete fog majd határt szabni a kontextus témájának és a kérdések típusainak. A kérdésgenerálásnak számos felhasználási területe van, mint például az oktatás, ahol megkönnyítheti a tanárok dolgát dolgozatok készítésekor, vagy akár a hallgatóknak is lehet vele készíteni segédanyagokat. Számos vállalati felhasználása is lehet, mint például felvételi tesztkérdések generálása, de lehet használni ezen megoldásokat akár egy chatbotban is, ahol más NLP feladatokkal együtt fel lehet mérni igényeket vagy beszélgetéseket lehet kezdeményeznie a chatbotnak is. A következő alfejezetben bővebben is kitérek a kérdésgenerálásra, annak módszereivel, típusaival és konkrét felhasználási területeivel együtt.

2.4. Kérdésgenerálás

A kérdésgenerálás feladatköre amellett, hogy hasznos segédeszközök létrehozását segítheti elő, remek mérőeszköze is lehet a mesterséges intelligenciának. Korábban a számítógépekkel történő kommunikáció kimerült abban, hogy különböző parancsokat adtunk a gépnek és azokat igyekezett a lehető legpontosabban elvégezni számunkra. Ez a működés amellett, hogy az ember számára meglehetősen kényelmes, egyben biztosította arról is, hogy akinek a parancsokat megadta, az egy gép és nem ember, hiszen valós, emberi kommunikáció nem történt, hanem csak egy gomb lett megnyomva, egy formális nyelven írt parancs lett megadva vagy egyéb automatika indította el a program futását.

A kérdésgenerálás azonban felbonthatja ezt a korábbi működést és a gépet helyezheti a kérdező pozíciójába. Képzeljünk el például egy ügyfélszolgálati chatbotot, aki felkeres minket valamilyen ügyben és értelmes kérdéseket generálva képes egy adott vállalatnál az ügyfél érdekeit képviselve elvégezni valamilyen adategyeztetési vagy üzleti feladatot. A chatbot ekkor azzal, hogy képes volt adott kontextusban kérdéseket generálni és képes volt az ügyfél által megadott adatokkal kapcsolatban kérdezni egyértelmű tanúbizonyságát adta, hogy intelligens, megértette feladatát és szeretné pontosítani az

ügyfél adatait és egyben saját feladatának elvégzését is igyekszik javítani. Ez természetesen még nem jelenti azt, hogy tudatra ébredt a gép, de a felhasználó itt már érezheti azt, hogy intelligens, emberszerű lénnyel beszél.

2.4.1. Kérdésgeneráló alkalmazások

Magának a feladatnak a formalizálása hosszú időre nyúlik vissza. Az első eredmények a területen a matematikai logika kérdésekre való alkalmazásának idejére tehető. Már 1929-ben **Cohen** által elkezdődött a kérdések vizsgálata, aki a kérdések tartalmát egy nyitott formulával és egy vagy több kötetlen változóval próbálta leírni[6], azonban a kérdések automatikus generálása mégis egy még frissnek mondható témakör.

1976-ban **Wolfe** készített egy tanulást segítő automatikus kérdésgeneráló rendszert AUTOQUEST néven.[7] Itt a kérdések a diákoknak kiadott szöveges anyagokból generálódtak, tehát már itt is körvonalazódott ezen rendszerek alapvető működése. A rendszer képes volt a megadott szövegekből szintaktikai és szemantikai információkat kinyerni. A szöveg szavait háromféle kategóriába sorolta(főnevek, igék és elöljárószavak), melyek segítségével meg tudta határozni a megfelelő kérdő névmást a generált kérdésekhez. A tesztek során a generált kérdések 80%-a bizonyult helyesnek és értelmesnek, míg szintaktikailag helyesnek a kérdések 93%-a volt tekinthető.

Mostow és **Chen** 2009-ben kifejlesztett egy olvasási oktató programot diákoknak, mely automatikus kérdésgenerálást használt, hogy javítsa a tanulók szövegértési képességeit.[8] A program alapvető célja, hogy ösztönözze a tanulókat kérdések feltevésére az olvasott szövegekkel kapcsolatban. Ezt úgy éri el, hogy először mutat egy példa kérdést számukra és utána adott választási lehetőségek alapján segít nekik kérdéseket konstruálni az olvasott szövegből. Ha a diák helyes kérdést tett fel, akkor a program pozitív választ ad, míg helytelen esetben új kérdést kell megadni neki. A tesztek során kiderült, hogy a kérdések csupán 35.6%-a bizonyult helyesnek, azonban a helytelen kérdések detektálásának pontossága 90% volt, ami nagyon jó eredménynek számított.

A korábbi szöveges kontextus alapú megközelítésekkel szemben 2013-ban **Jouault** és **Seta** egy szemantikai alapú kérdésgeneráló programot készített, ami a Wikipedia adatbázisából lekérdezett szemantikai információk alapján képes kérdéseket generálni.[9] A felhasználók feladata, hogy egy adott dokumentumból felépítsenek egy idővonalat és felvegyék az egyes események közötti relációkat, tehát egyfajta koncepció vagy szemantikai térképet kell készíteniük. Eközben a program is elkészíti saját koncepciótérképét a felhasználó térképe, illetve a Wikipedia szemantikai információi alapján. Ezt felhasználva a program később kérdéseket tud generálni a felhasználó számára, melyekkel elmélyítheti tudását az adott témában.[6]

2.4.2. Korábbi megközelítések

Számos megközelítés létezik egy kérdésgenerálási feladat megoldására, azonban ezen megoldásokban több közös lépés is van. Az egyik ilyen lépés az előfeldolgozási fázis, ahol akárcsak a többi NLP feladat esetében a bemenetet a programunk számára kedvező formátumba alakítjuk. Itt lehetőségünk van a szkript számára felesleges szavakat vagy karaktereket kiszűrni és egyszerűsíteni a bemeneten. Egy másik ilyen gyakran ismétlődő lépés, hogy a programnak fel kell tudnia ismernie különböző nyelvtani szerkezeteket, úgy mint főneveket, igéket, névmásokat stb. és ezeket megfelelően csoportosítva értelmeznie kell a mondatot, majd pedig akár ezen kiszűrt lényeges entitások felhasználásával is meg kell konstruálnia az egyes kérdéseket. Utolsó lépésként pedig elő kell állítania a kimenetet általunk is értelmezhető formátumban, természetes vagy formális nyelven, kérdésekként.

Amiben viszont eltérhetnek ezen megoldások, az a közbenső legfőbb lépés, ahol magukat a nyelvtanilag helyes, természetes nyelvű kérdéseket generáljuk.

Az első megközelítés az az **átalakítás alapú** kérdés generálás. Ennek során az algoritmus először azonosítja és törli az adott célfogalmat a szövegben, megkeresi a megfelelő kérdéstípust és a hozzá tartozó kérdő névmást, majd pedig az igét nyelvtanilag helyes formára alakítja a kiegészítő és modális igék segítségével. Például a *"10 óráig van nyitva a bolt."* mondathoz felismeri az algoritmus az időpontot, leválasztja az időpontra vonatkozó célfogalmat, a kérdés elejére beszúrja a *"Meddig"* kérdő névmást, majd végül már csak át kell alakítania a mondat végi írásjelet és kész is a *"Meddig van nyitva a bolt?"* kérdés. Ez a megközelítés működhet, azonban csak úgy, mint a többi NLP feladat esetén itt is előjön, hogy ha ismeretlen mondattípussal találkozik a rendszer vagy szimplán rosszul megfogalmazott a bemeneti szöveg, esetleg hibás, akkor értelmetlen kérdések fognak generálódni.

Egy másik hasonló megoldás a **sablon alapú(template-based)** kérdésgenerálás. Itt az az alapötlet, hogy minden kérdéstípushoz létre lehet hozni kontextusspecifikus kérdés sablonokat, melyek a legjobb esetben le fogják fedni az összes kérdésosztályt. Ilyen sablon lehet például a *Mi lenne, ha <X>?* a feltételes módú szövegeknél, vagy a *Mi történik <X>?* és *Mikor lenne <X>?* időbeli kontextusok esetén, ahol az *<X>* változót az algoritmus különböző szemantikai szabályok alapján helyettesíti be. Ez a megoldás elsősorban ismeretközlő szövegek esetében működik, ahol nagyon kötött a szórend és egyértelműen meg lehet találni az egyes változókat. Elbeszélő szövegek esetén szükség lehet egyéb reguláris kifejezés alapú szabályok bevezetésére, melyek segítségével az algoritmus kinyerheti a szükséges információkat a szövegekből. A feladat megközelítése ebben az esetben is eredményes volt, azonban a program feldolgozható témaköreinek növelésével a szükséges szabályhalmaz jelentősen nő és romlik az algo-

ritmus sikeressége is, így ezen megoldásokat leginkább specifikus, zártabb esetekben alkalmazzák.

Napjaink legfejlettebb megoldásai azonban kétségkívül a neurális hálózat alapú mély tanulást alkalmazó algoritmusok. Az ilyen típusú megoldások esetében általánosan elmondható, hogy a fejlesztő nem vesz fel hatalmas méretű szabályhalmazokat a feladat elvégzéséhez, hanem helyette kialakít egy olyan neurális hálózat modellt, mely magától képes ezeket a szabályokat felismerni és ezáltal oldja meg a különböző NLP feladatokat. A következő fejezetben ezen modernebb módszereket fogom bemutatni előnyeikkel és hátrányaikkal együtt.

3. fejezet

Modern módszerek a nyelvfeldolgozásban

Az utóbbi idők legjelentősebb fejlődését a területen a neurális hálózatok megjelenése indukálta. Korábban számos próbálkozás született szabályok, sablonok, statisztikai megoldások felhasználásával, azonban ezek például egy chatbot vagy szöveggenerálási feladat esetén hamar problémákba ütköztek, hiszen egy újabb, addig ismeretlen nyelvi elem vagy egy speciálisabb kontextus teljesen meg tudta állítani ezen programok működését. További probléma volt, hogy mivel magát a nyelvet, annak kódolását és dekódolását is emberek találták ki emberi aggyal, így egyszerű képletekkel, szabálybázisokkal ez a probléma nem volt megoldható, mivel az emberi agy - mint az számos kutatásból kiderült - nem használja beépítve ezeket a működéseket, például nem kezdi el egyenként, szekvenciálisan feldolgozni a betűket, hanem belső állapotától függően képes egyben látni szavakat, mondatokat és leginkább predikciókkal dolgozik, mint sem pontosan leírt, kötött műveletekkel, szabályokkal.

3.1. Neurális hálózatok

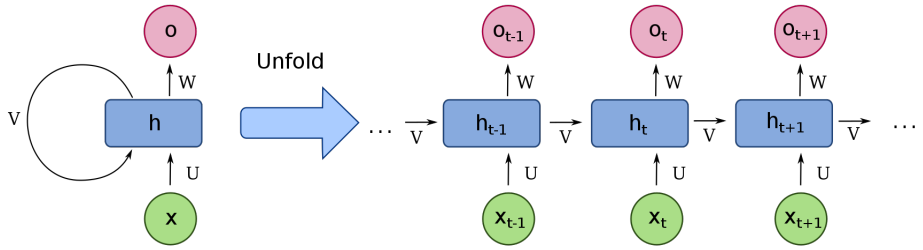
Mindezen problémák miatt logikus lépés volt megvizsgálni az emberi agyat és előállni egy olyan koncepcióval, mely képes modellezni az agy mintafelismerési képességét és ezáltal új távlatokat nyitni a nyelvfeldolgozás területén. Ezek voltak a **neurális hálózatok**.

Az idők során számos neurális hálózat típus jelent meg: perceptron, feed forward, MLP, konvolúciós, visszacsatolt(RNN) stb. Ezek közül számunkra a nyelvfeldolgozás területén a legfontosabb és legtöbbet használt típus az RNN(Recurrent Neural Network).

A neurális hálózatok felépítése nagyon változatos lehet, azonban számos közös jel-

lemzőjük akad. Minden hálózatnak tartalmaznia kell egy bemeneti réteget, egy rejtett réteget és egy kimeneti réteget. Ezen rétegek jellege a feladat típusától függően változtatható, például a hálózat bemenete lehet többdimenziós, a rejtett rétegekben változtathatjuk a neuronok kötéseit, illetve az adatmozgások irányát, a kimenete pedig lehet egy skalár mennyiség vagy egy vektor, attól függően, hogy osztályozni vagy regressziót számítani szeretnénk.

RNN esetén a feladat egy szekvencia feldolgozása, ami lehet egy kép, amit szeretnénk felcímkézni, hangfájlok, amik segítségével beszédet szeretnénk felismerni, illetve akár egy szöveg, amit szeretnénk lefordítani vagy értelmezni. Ami közös ezekben a feladatokban, hogy mindegyik problémakör esetén a feldolgozás során a hálózat bemenete és kimenete jelentősen függ egymástól, vagyis a szekvencia egyes elemeinek értelmezése függ a korábbi elemek értelmezésétől.



3.1. ábra. Az RNN működése [10].

Kiváló példa erre a szövegfordítás, hiszen a fordítás során fontos a szavak rendje, tehát a hálózatnak sorban, egymás után kell vennie a forrásszöveg szavait és kimenetén ezen szavak adott nyelvű megfelelőjének kell megjelennie.

Ez a működés jelentős előrelépés volt, azonban számos probléma felmerült ennek kapcsán. Az egyik ilyen probléma a hosszabb szövegek értelmezése volt. Ilyenkor a hálózat egyszerűen elfelejtette azt a tudást, amit a szöveg értelmezésének elején megszerzett, ezt hívjuk "vanishing gradients" problémának. Ennek magyarázata a hibafüggvény gradiensében keresendő, ami nem más, mint a hibafüggvény deriváltja a hibagörbe mentén. Amikor ez a gradiens túl kicsi, akkor idővel még kisebbé válik és ezekkel az alacsony, nagyon 0-hoz közeli értékkellel kezdi el frissíteni a hálózat súlyait, egészen addig, amíg azok le nem nullázódnak. Ebben az esetben a hálózat nem tanul tovább. Ennek a problémának létezik a fordítottja is, az "exploding gradients", melynek során a gradiens túl nagy lesz, ezáltal egy instabil modellt alkotva, melynek hatására a súlyok túl nagyok és idővel NaN értékűek lesznek.

Ezen problémákra születtek megoldások, például a hálózat komplexitásának csökkentése, vagyis a rejtett rétegek számának redukálása, azonban ez nem mindig vezet

optimális megoldásra.

Egy másik probléma az RNN-el, hogy a hálózat szekvenciális feldolgozásra készült mivoltából adódóan egyszerűen nem jól párhuzamosítható, vagyis a mai modern hardverekkel, például egy rengeteg, erőteljes párhuzamosításra használható maggal felszerelt GPU-n nem tudjuk effektíven tanítani a hálózatot, ami a nagyobb szövegek értelmezését rettentően időigényessé teszi.

3.2. Transformer hálózatok

Az RNN tehát minden problémájának ellenére is hatalmas sikereket ért el az NLP területén, azonban 2017-ben egy új neurális hálózat típus jelent meg:
az átalakító(transformer), ami a fentebb említett problémákat javítva és a fejlesztést is egyszerűbbé téve átvette a vezetést a szövegfeldolgozás területén.

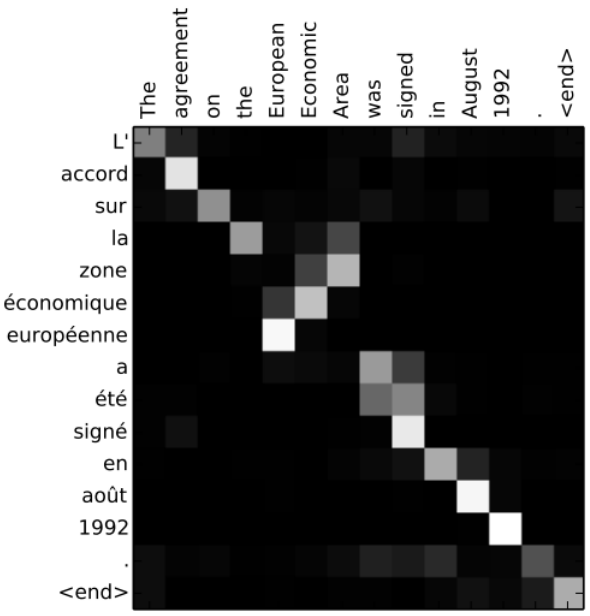
Az átalakítókat a Google és a Torontói Egyetem szakemberei fejlesztették ki 2017-ben és meg is jelentettek egy cikket "Attention Is All You Need"[11] címmel, amiben részletezték ezen átalakítók működését. Talán a legnagyobb újítás a párhuzamosíthatóság területén jelent meg, mivel ezeket a hálózatokat a megfelelő eszközökkel hatalmas mennyiségű adatokkal lehet tanítani. Például a Google T5 nevű átalakító modelljének a többnyelvű változatát a *c4/multilingual* nevű adathalmazzal tanították, ami 26.76 TiB méretű(1 TiB = 1.1 TB). Később az OpenAI vállalat GPT-3 nevű modellje még ezt is túlszárnyalta szinte a teljes publikus internetet tartalmazó 45 TB méretű szöveges adatot tartalmazó tanítóadatával. Ezen méretű tanítóhalmaz korábban elképzelhetetlen volt az RNN-ek használatával.
Az átalakítók működését 3 fontos fejlesztésre lehet lebontani:

- Pozíciókódolások(Positional Encodings)
- Figyelem(Attention)
- Önfigyelem(Self-Attention)

A **pozíciókódolással** a korábbi RNN-ek szekvencialitását oldották fel azáltal, hogy a mondatokat nem a szavak sorrendjében kezdték el feldolgozni, hanem a mondat minden szavát ellátták egy annak mondatban elfoglalt pozícióját jelentő címkével. Ez a struktúra szó-sorszám párokat jelent, amiket a hálózat megtanul hatásosan használni. Ennek segítségével a szavak sorrendje immáron nem a hálózat struktúráját jelenti, vagyis a szekvencialitást, hanem egyszerű feature-nek, adatnak tekinthető.

A **figyelem** egy olyan mechanizmus, melynek segítségével a modell végigmehet a bemenet minden szaván és megadhatja egy szónak a jelentését az alapján, hogy melyik

ismert idegennyelvű szóhóz áll a legközelebb a szintaktikája. Ezt a tudást a tanítás során szerzi meg a modell, ezért is van szükség minnél nagyobb adathalmazokra. Ez a működés elsősorban a modell célterületén vagyis szövegfordítások esetén hasznos, ahol egy adott nyelvű mondat fordításánál a szórend változhat, és nem elég szimplán az egyes szavakat lefordítani, hanem szükség van egyfajta háttértudásra, nyelvi ismeretekre a fordítás során. Pélául a *"The agreement on the European Economic Area was signed in August 1992."* angol nyelvű mondat francia fordítása *"L'accord sur la zone économique européenne a été signé en août 1992."* Láthatjuk, hogy a *"European Economic Area"* fordítása *"la zone économique européenne"*, tehát a szórend és a szavak alakja is változik. Ebben az esetben nem elég az egyes szavakat szekvenciálisan fordítani, hanem minden angol szóhoz a forrásmondatban fel kell építeni egyfajta hőtérképet a francia fordításokkal és a modellnek végig kell néznie az egyes szavakat és meg kell mondania, hogy melyik angol szóhoz melyik francia szó illeszkedik. Esetünkben például az *"European"* szóhoz illeszkedik az *"européenne"* és az *"économique"* szó is, azonban a modell korábbi tanításából adódóan tudja, hogy itt az *"européenne"* fordítás lesz a helyes.



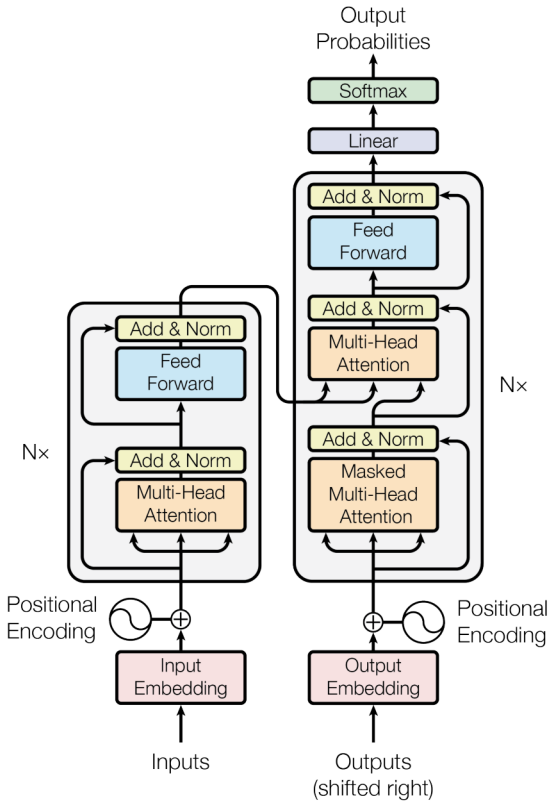
3.2. ábra. Angol-francia fordítás hőtérképe [12].

Az utolsó fontosabb fejlesztés az **önfigyelem**, ami talán a legfontosabb a szövegértelmezés szempontjából. Korábban láthattuk, hogy a figyelem segítségével a modell képes megfelelő sorrenben fordítani a szavakat, azonban ez még nem elég ahhoz, hogy képes legyen érteni is az egyes szavak jelentését és ezáltal más szövegfeldolgozási feladatokat is meg tudjon oldani. Ennek érdekében szükségessé vált, hogy a modell mögött

álló neurális hálózat felépítsen egy belső reprezentációt az adott nyelvről. Ez a belső működés leginkább a különböző képfelismerési hálózatok(CNN) rétegeihez hasonló, ahol az egyes régetek képesek felismerni éleket, alakzatok és egyéb magasabb szintű, komplexebb struktúrákat, mint emberek, állatok vagy tárgyak. Nyelvi környezetben ezen rétegek képesek felismerni a különböző nyelvtani szabályokat, szinonímákat és szövegrészleteket. A célja ezen rétegeknek, hogy minnél jobban megtanulják az egyes nyelvtani elemeket és kontextusokat, így a modell képes lesz szinte bármilyen nyelvi feladatot megoldani.
Vegyük példának a következő angol nyelvű mondatokat:

1. *"Server, can I have the check?"*
2. *"Looks like I just crashed the server."*

A *"server"* szó ebben a 2 mondatban 2 különböző jelentéssel bír: az egyik mint felszolgáló vagy pincér, míg a másik egy webes kiszolgálóra utal. Mi emberek a *"server"* szó körül lévő szavakból könnyen meg tudjuk különböztetni a 2 jelentést, azonban ez a gépeknek korábban nem volt egyszerű feladat. Az önfigyelem erre a feladatra nyújt megoldást azáltal, hogy képes az egyes szavakat más szavakhoz kötni és így a modell megtanulja, hogy abban az adott kontextusban mit is jelenthet az a szó. Például az 1. mondatban a *"server"* szó mellett megtalálható a *"check"*, a *"can"* és a *"have"* szó is, melyek együtt sűrűbben szerepelnek egy éttermi szituációt leíró kontextusban, mint a webfejlesztés esetében, tehát itt egy pincért jelent a szó, míg a 2. mondatban megtalálható a *"crashed"* szó, ami pedig az informatikában és webes környezetekben gyakoribb, ezáltal a *"server"* itt egy webkiszolgálót jelent.



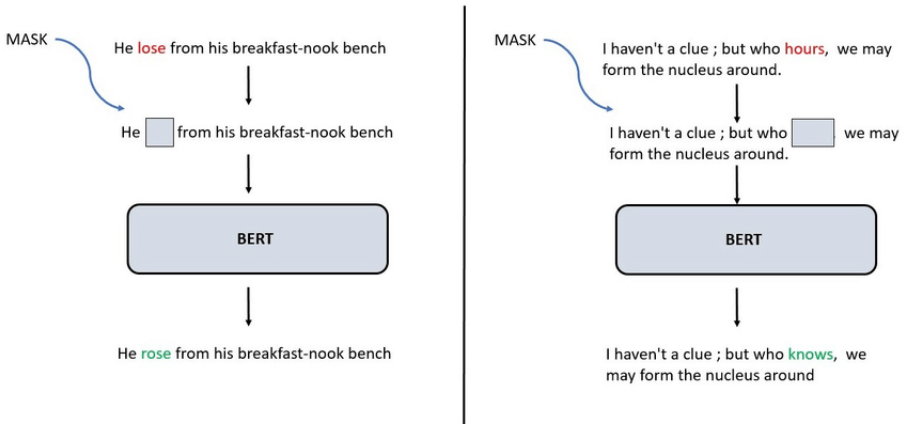
3.3. ábra. Az átalakító modell architektúrája [11].

3.3. A BERT modell

Ez a 3 fontosabb fejlesztés indítottal el útjára az átalakító alapú modelleket, melyek közül az első jelentősebb a Google által 2018-ban kifejlesztett **Bidirectional Encoder Representations from Transformers(BERT)** volt. A BERT nem csak egy újfajta modell architektúra volt, hanem egy teljesen új betanított modell, amit ingyenesen letölthetővé is tettek. Kisebb átalakításokkal számos probléma megoldására képes volt: szövegösszefoglalás, kérdés-válasz generálás, osztályozás és még sok más feladat. Működésének legfontosabb fejlesztése az átalakító modell kétirányú kiterjesztése volt. Korábban az átalakító modellek a tanítás során balról jobbra vagy kombináltan balról jobbra és jobbról balra dolgozták fel a szövegeket. Ezzel szemben a BERT a szavak értelmezésénél a környező szavakat mind a két lehetséges irányban egyszerre dolgozza fel, ami a szövegek mélyebb megértését teszi lehetővé.
Ahhoz, hogy ez a működés megvalósuljon 2 fajta tanítási stratégiát használ a BERT:

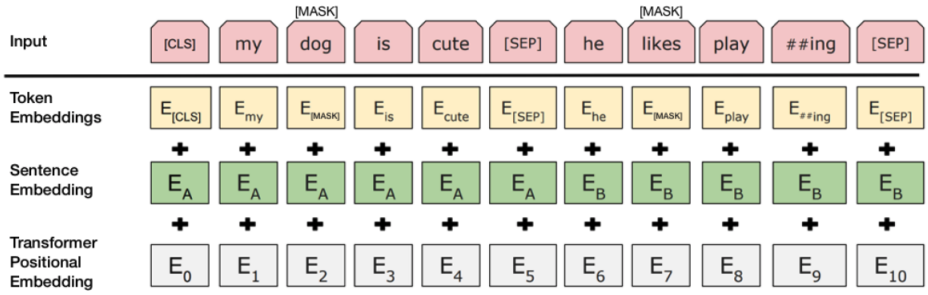
- Masked-Language Modeling (MLM)
- Next Sentence Prediction (NSP)

A **Masked-Language Modeling** az adott szöveg, pontosabban a szöveg szavainak mélyebb megértését célozza. A BERT hálózat tanítása során az egyes mondatok szavainak kb. 15%-át kicserélik [MASK] tokenekre. Ezt követően a modell megpróbálja kitalálni ezeket a maszkolt szavakat a körülötte lévő nem maszkolt szavak kontextusa alapján. A predikció során a maszkolt szavak mindkét oldaláról figyelembe veszi a nem maszkolt szavak kontextusát, innen ered a kétirányúsága a modellnek. Ez a működés nagyon hasonlít ahhoz, ahogy mi emberek értelmezünk egy szöveget vagy próbáljuk kitalálni egy ilyen feladat során a hiányzó szavakat.



3.4. ábra. Maszkolt szavak beillesztése a BERT működése során [13].

A **Next Sentence Prediction** esetén az MLM-el szemben nem a szöveg szavainak a megértése a cél, hanem az egyes mondatok közötti kapcsolatok feltárása. Ennek érdekében a tanítás során a modell mondat párokat kap, melyek második eleméről el kell döntenie, hogy az első mondat után következnek-e az eredeti forrásszövegben. A gyakorlatban a bemeneti szöveg mondatainak 50%-a olyan páros, ahol a mondatok egymás után következnek, a másik 50%-a pedig olyan, ahol a második mondat random kerül kiválasztásra és feltesszük, hogy a random mondat nem lesz kapcsolatban az első mondattal.



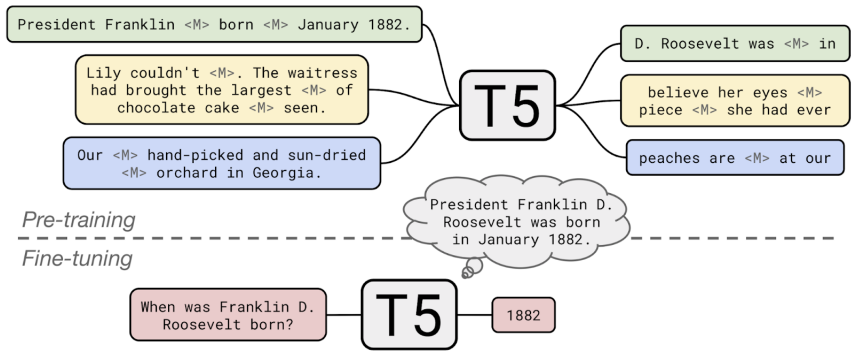
3.5. ábra. Next Sentence Prediction a gyakorlatban [14].

3.4. A T5 modell

A BERT jelentős sikerei után elkezdődtek a kutatások a modell felhasználói igényekhez történő igazítására, egyszerűsítésére, adathalmazának kibővítésére és tisztítására, illetve a modell képességeinek újabb feladatokra történő kiterjesztésére. 2020-ban a Google kutatói elő is álltak a **T5** nevezetű modellel. A modell fő célja az volt, hogy a korábbi fejlesztésekkel ellentétben a modell bemeneti forrásszövege és a kimenet is egységes szöveges formátumú legyen minden NLP feladat esetén.

A modell előtanítása során a **Colossal Clean Crawled Corpus(C4)** nevű adathalmazt használták, ami közel 700 GB méretű és a Common Crawl adathalmaz egy tisztított verziója. A C4 adathalmaznak létezik többnyelvű változata is az **mC4**, amely már tartalmazza a magyar nyelvet is sok más nyelv mellett. Az mC4 adathalmazon tanított T5 pedig az **mT5** nevet viseli és képes magyar nyelvű szövegek generálására is.

Belső működésében a T5 hasonlóan működik mint a BERT. A Masked-Language Modeling ugyanúgy megmaradt, azonban kibővítették azzal, hogy immáron nem csak egy-egy szót, hanem egyszerre több egymás melletti szót is maszkol, amit a modellnek ugyanúgy ki kell majd találnia. Ennek érdekében a forrásszöveget bemenet-cél párosokra bontja és ezeket fogja megtanulni a tanítás során. A BERT-el ellentétben itt a kimenet nem egyetlen vektor lesz, hanem egy generált szöveg, tetszőleges mérettel.



3.6. ábra. T5 modell a tanítás előtt és után [15].

Az előtanítás után a modellt finomhangolták számos NLP feladatra: fordítás, összegzés, mondathasonlóság stb. A finomhangolás során bevezettek egy egyedi formátumot a különböző feladatok különválasztására. A forrásszöveg elé beszúrtak egy prefixet, ami az adott NLP feladatot jelöli. Ennek formátuma:

feladat_ azonosítója: forrászöveg

Ez azért volt szükséges, hogy a modell súlyait feladatok szerint tudják csoportosítani és így az egyes feladatokra való finomhangolás nem zavar bele a többi feladat megoldásába. További pozitívum a modellel kapcsolatban, hogy teljesen ingyenesen

hozzáférhető és felhasználható, így dolgozatomban is ezt a transformer modellt használtam a kérdésgeneráló alkalmazás elkészítéséhez.

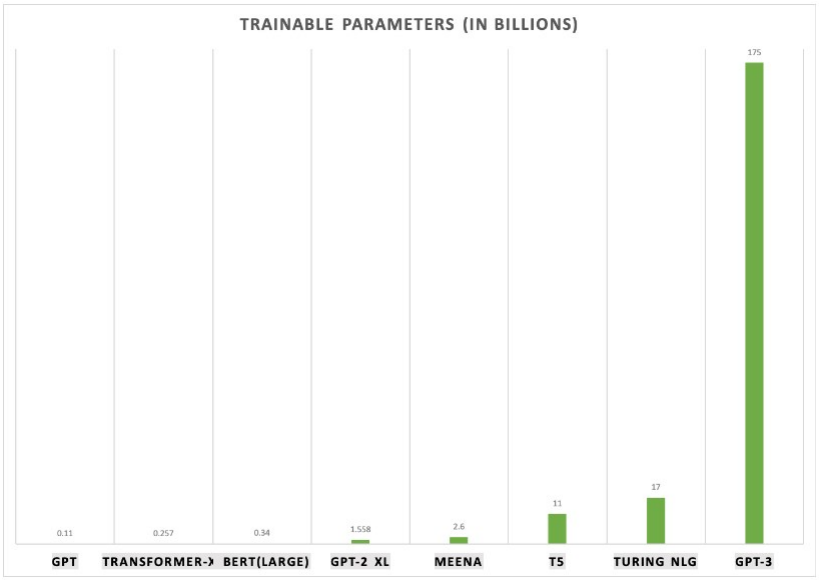
3.5. A GPT-3 modell

Napjaink egyik legsikeresebb és legelterjedtebb NLP modellje kétség kívül a **GPT-3**. A modell, melyre a ChatGPT nevű chatbot is épül jelentősen felforgatta az NLP teljes területét. Már a a konkrét alkalmazásának megjelenése előtt is óriási elvárásokat támasztottak vele szemben a szakterület kutatói és mára nyugodtan elmondhatjuk, hogy szinte mindegyiknek sikeresen megfelelt.

Láthattuk a korábbi modelleknél is, hogy folyamatosan cél volt a modell fejlesztésének egyszerűsítése. Míg a BERT-nél a visszatérési értékünk egy a neurális hálózatoknál gyakori vektor volt, addig a T5 modell már konkrét és könnyen értelmezhető szöveges eredményeket produkált. A GPT-3 és korábbi verziói ezen a területen igyekeztek javítani elsősorban és elvetették a T5 által is még erősen alkalmazott feladat specifikus működést és egy sokkal általánosabb megközelítést kezdtek el használni, melynek első lépése maga a modell tanítása volt.

Az első dolog, ami a GPT-3 vizsgálata során feltűnő az a modell tanulási paramétereinek a mennyisége. A tanulási paraméterek száma nem más mint a neurális hálózat súlyainak és az ún. *bias* értékeknek az összege, vagyis egyenesen arányos a hálózat méretével, csomópontjainak számával. A GPT-3 esetén ez az érték tízszer nagyobb, mint a korábbi modellek esetében. Természetesen a modell méretének növelésével a tanítási adatok mennyiségét is jelentősen növelni kell. A modellt hatalmas nagy méretű szöveges adatforrásokon tanították. Az alkotók szerint[16] körülbelül 45 TB méretű, több forrásból származó szöveges adatot használtak fel a tanításhoz:

GPT-3 adatforrások		
Adathalmaz	**Méret(tokenek)**	**Súlya a tanulásban**
Common Crawl	410 milliárd	60%
WebText2	19 milliárd	22%
Books1	12 milliárd	8%
Books2	55 milliárd	8%
Wikipedia	3 milliárd	3%



3.7. ábra. A GPT-3 és a korábbi modellek tanulási paramétereinek száma.

A modell tanítása után a készítők elkezdték tesztelni a modellt különféle módokon és megfigyelték, hogy nagyon hamar képes magától, felügyelet nélkül tanulni. Néhány példa alapján könnyen betaníthatóvá vált a modell mondjuk hangulatelemzésre, kérdés megválaszolásra vagy szintaktikai elemzésre, javításra. A kutatók a működést **"in-context learning"**-nek nevezték el[16], vagyis egyfajta kontextusban való tanulásnak, melynek során a hálózat súlyai nem frissülnek és nincs szükség a hálózat előtanítására, mégis képes megoldani a feladatokat. A jelenség magyarázata az lehet, hogy a hálózat ilyenkor pontosítani próbálja a kontextust a megadott példákkal és a Bayes-i inferenciához hasonló módon az új információk alapján próbálja meg pontosítani a válaszát.[17] Ez a működés lehetővé tette a GPT modellek számára, hogy általános célúvá váljanak és ne csak egy-egy témakörben vagy feladatkörben tudjanak működni. A modell hátránya, hogy sajnos a kisebb GPT-2-es verzión kívül nem ingyenes hozzáférésű, így csak az OpenAI vállalat platformjain lehet, regisztráció után hozzáférni.

4. fejezet

A kérdésgeneráló webalkalmazás megvalósítása

Ezen fejezet célja az általam készített kérdésgeneráló mintaalkalmazás bemutatása. Terítékre kerül az alkalmazás célkitűzése, tervezése, fejlesztési menete, végül pedig az alkalmazás tesztelése, mind automatikus, mind pedig manuális formában. Kitérek továbbá a felhasznált programozási eszköztárak bemutatására, azok előnyeivel és hátrányaival együtt.

4.1. Az alkalmazás célja

Ez a webalkalmazás a dolgozatomban bemutatott módszerek gyakorlati alkalmazásának bemutatására szolgál. Az alkalmazás alapvetően egy online kérdésgenerátor, melynek meg lehet adni egy szöveges kontextust és az alapján a program mögött álló modell generál bizonyos számú kérdést. A célom az volt, hogy biztosítsak egy nagyon egyszerű, átlátható és jól kezelhető interfészt, továbbá hogy maga az algoritmus is minél relevánsabb kérdéseket alkosson meg. Mindezt online formában képzeltem el, hiszen az alkalmazás szerver oldali(backend) része meglehetősen erőforrás-igényes, így azt egy egyszerű asztali alkalmazás formájában nehéz lenne futtatni, illetve így bárhonnan szabadon elérhetővé válik a program.

4.2. A programmal szemben támasztott követelmények

- **Gyors válaszidő, optimális működés**: Mivel az alkalmazás mögött egy nagyobb neurális hálózat van, így mindenképp biztosítani kell, hogy a szerver ren-

delkezzen a megfelelő mennyiségű memóriával, processzormaggal vagy akár videokártyával, annak érdekében, hogy reális válaszidőket kapjon a felhasználó. Továbbá a kódnak is alkalmazkodnia kell a körülményekhez, így rövidnek, átláthatónak és gyorsnak kell lennie.

- **Letisztult felhasználói interfész**: Az alkalmazás alapvető céljából fakadóan biztosítani kell az egyszerű kezelhetőséget. Alkalmasnak kell lennie tetszőleges méretű szöveges kontextus feldolgozására, illetve a generált kérdéseknél is törekedni kell a könnyű másolhatóságra, hogy a felhasználó később be tudja illeszteni dokumentumaiba őket.

- **Megfelelő minőségű kérdések**: Az alkalmazás által kreált kérdések nem térhetnek el a megadott kontextustól, de törekedni kell arra, hogy ne csak a megadott szövegből kimásolva kérdezzen, hanem érződjön valamilyen háttértudás is a kérdések mögött. Illetve a kérdéseknek az adott nyelven értelmesnek kell lenniük.

- **Általános tudásbázis**: Szükség van arra, hogy az alkalmazás minél több témakörből képes legyen kérdéseket generálni, így olyan tanítóhalmazra van szükség, ami nem korlátozódik egyetlen témára, illetve több kérdés típust is tartalmaz.

4.3. A program megtervezése

Az alkalmazás elkészítését hosszas tervezőfolyamat előzte meg. Első lépésként választani kellett egy megközelítési módot az alapfeladat megvalósításához. Számos módszertan létezik a kérdésgeneráláshoz, de végül a neurális hálózat alapú megoldás mellett döntöttünk, azon belül is a már részben előtanított transformer modellek mellett, melyek már rendelkeznek alapvető háttértudással, így már csak a kérdésgenerálás megvalósítására kell megtanítani őket.

Következő lépésként el kellett dönteni, hogy milyen nyelven történjen a kérdésgenerálás. Itt szükséges volt összeszedni, hogy az egyes nyelveken milyen mennyiségben áll rendelkezésre tanítóhalmaz a neurális hálózat tanításához, hiszen elengedhetetlen, hogy a hálózat lásson kontextusokat és hozzájuk készült kérdéseket mintának, ami alapján később képes lesz saját magától is hasonló kérdéseket alkotni. Végül az angol nyelvre esett a választás, mert ezen a nyelven találtunk megfelelő tanítóhalmazt és a feladathoz illő alaptudással rendelkező hálózatot, melyet később tovább tudtunk tanítani.

Ezután a megfelelő transformer hálózat kiválasztása következett. Számos modellt megvizsgáltunk az alapján, hogy:

- Mennyire megoldható rajta feladatunk?

- Mennyire optimális a működése?
- Tudjuk-e saját gépen vagy egyéb szerveren futtatni, tárolni és tanítani?
- Mekkora a mérete?
- Kellően fejlett-e a feladat megoldásához?
- Ingyenesen elérhető-e és lehet-e tovább tanítani?

Először a BERT modellt vizsgáltuk. Számos kutatást és cikket találtunk a témában, melyek sikeresen alkalmazták a BERT-et több NLP feladatra. Kérdésgenerálásra is alkalmazhatónak tűnt, azonban a tanítása nehézkesnek bizonyult, illetve nagyon hamar találtunk sokkal jobb és kifejezetten szöveggenerálásra használható modelleket, melyek könnyebben kezelhetők, gyorsabbak és jobb eredményeket is produkáltak.

A GPT-2 nevű modell már ígéretesebbnek tűnt, hiszen kifejezetten szöveggenerálásra hozták létre és kb. 40 GB szöveges adattal tanították be, ami már jelentős eredménynek számított a területen. Ingyenesen elérhető és letölthető, valamint szabadon lehet tovább tanítani is. Szöveggenerálási eredményei elég jók voltak és számos kutatást találtunk, ahol kérdés megválaszolásra és kérdés generálásra tanítják, de más modellekkel összehasonlítva nem ezt találtuk a legjobbnak.

A GPT-3 modell minden szempontból túlszárnyalta elődjét. Amellett, hogy a hálózat architektúráját átszervezték, jelentősen növelték a tanulási paramétereinek számát és a tanítási adathalmaz méretét is, így rendkívül jó eredményeket produkál szinte az összes NLP feladat esetén, így a kérdésgenerálásban is. Egyetlen hátránya a modellnek a mi szempontunkból, hogy csak korlátozottan használható és jelenleg nem ingyenes a hozzáférése.

Számunkra a legmegfelelőbb modellnek a Google T5 nevezetű modellje bizonyult, ami már kifejezetten fejlett, kiválóan alkalmazható mindenféle szöveggenerálási feladatra és nagyon könnyen betanítható kérdésgenerálásra, amellett, hogy meglehetősen kedvező méretű. A Hugging Face nevű oldalon ingyenesen elérhető módon meg is található betanított változatban, melyet tovább tanítva megoldhattuk feladatunkat.

Mivel az alkalmazás céljai között szerepel, hogy online elérhető legyen, így ezután elkezdődhetett a szerveroldali működés kidolgozása. A szerver programozási nyelvénél a Python-ra esett a választás, mivel nagyon hasznos könyvtárai vannak adatelemzés és statisztika területén, illetve webfejlesztési területen elérhető hozzá egy Django nevű szerveroldali keretrendszer is, mely jelentősen megkönnyítette a szerver elkészítését. Magának a szervernek egyetlen főbb végpontot kellet biztosítani, ami a megadott szöveges kontextust fogadja és válaszként visszaadja az elkészült kérdéseket. Miután az alkalmazás nem tárol felhasználói adatokat, így adatbázisszerver nem lett konfigurálva

hozzá, de a rendszer úgy lett kialakítva, hogy ha idővel mégis szükség lenne rá, akkor könnyen be lehet csatlakoztatni egy tetszőleges adatbázist.

Mindezek mellett szükségessé vált időközben, hogy magának a neurális hálózatnak a tanítását és működését valamilyen szinten szeparáljuk a szervertől, hiszen ez a feladatrész akár napokig is eltarthat még erősebb hardver esetén is. Ezért született az a döntés, hogy a tanításért felelős kódokat vegyük külön egy önmagában futtatható kódfájlba, melyet bárhol képesek vagyunk futtatni, például valamilyen felhőszolgáltatáson és utána a betanított hálózatot, ha van rá lehetőség mentsük el egy tárhelyre, majd onnan a szerver töltse be és már készen, előtanítva használja azt. A tanításért felelős felhőszolgáltatásnak a Google Colab nevű rendszerét használtuk, míg a betanított hálózatot a Hugging Face nevű oldalon tároltuk el, ahonnan egy egyszerű API-n keresztül könnyen le is tudtuk tölteni saját szerverünkre, ahol egyszerű fájlként tud tárolódni.

Miután megterveztük a szervert következhetett a kliensoldali(frontend) rész. Itt a Vue.js nevű frontend keretrendszert választottuk, mivel kisméretű, gyors és könnyű benne a fejlesztés is. A rendszer komponensalapú megközelítése segítette az alkalmazás egyszerű és átlátható működésének biztosítását. Az klienshez igyekeztünk letisztult dizájnt választani, melyen egyértelmű minden funkció és biztosított, hogy a felhasználó a program minden funkciójának tudja a szerepét. Szempont volt például, hogy tetszőleges méretű kontextust tudjon kezelni a kliens, illetve, hogy a kérdések együtt és külön-külön is kimásolhatóak legyenek a későbbi felhasználásukhoz. A felületen történő navigációt pedig ikonok és apró súgószövegek segítik.

Az utolsó lépés a tesztek és tesztesetek előkészítése volt. A jelenlegi és jövőbeli fejlesztés megkönnyítése érdekében szükséges volt automatizált teszteket készíteni, mint például egységteszteket és integrációs teszteket. Mindezeket kliens és szerveroldalon is el kellett készíteni. Továbbá, mivel egy NLP feladatot oldunk meg, így szükségessé vált a manuális, kézi tesztelés is különböző tesztesetekkel. Elsőkörben természetesen az alap eseteket kellett vizsgálni, hogy egyáltalán értelmes kérdések készülnek-e, majd pedig következhettek a komplexebb szituációkat bemutató tesztek.

4.4. Telepítés és futtatás

Mivel alkalmazásunk alapvetően webes környezetbe lett szánva, így lokális futtatása nem feltétlenül egyszerű feladat, de azért törekedtünk a könnyű elindítás biztosítására. Ennek érdekében elérhetővé tettük az alkalmazást Docker környezetben. A Docker használatához mindössze a Docker Desktop nevű alkalmazás telepítése szükséges, amely elérhető Windows, Linux és MacOS operációs rendszereken is. Ekkor a frontend és backend rész együttes futtatása mellet a http://localhost:3000 cím alatt válik elérhetővé

az alkalmazás.

Azonban a tesztelés megkönnyítése végett éles környezetben is elérhetővé tettük a webalkalmazást a következő URL alatt:

`https://mb-thesis-t5-qg-frontend-oumla56ewq-lm.a.run.app`

Az éles környezetet a Google Cloud biztosítja számunkra, ahol bizonyos mennyiségű havi szerver felé indított kérésig ingyenesen biztosítanak nekünk lehetőséget alkalmazásunk tárolására. A nap első indításánál lassabb lehet az oldal betöltése, de általában gyorsan el lehet érni a weboldalt és a kérdésgenerálás se tart túl sokáig a biztosított hardvereken.

Az alkalmazás lokális elindításához elengedhetetlen néhány alapkövetelmény, melyeknek rendszerünknek meg kell felelnie:

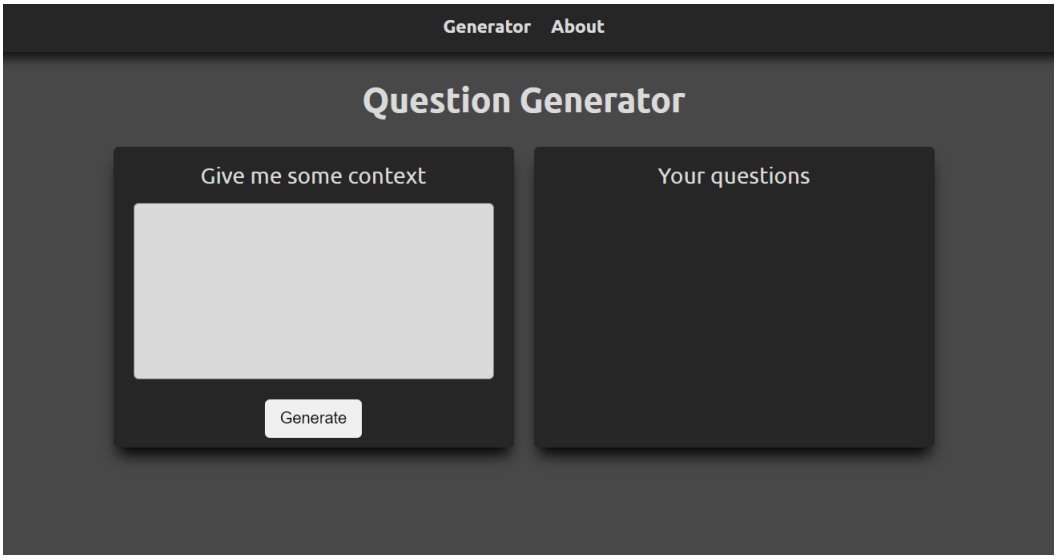
- Hardverek tekintetében bár igyekeztünk tehermentesíteni az alkalmazást a szeparált tanítókomponensekkel, de a szervernek így is jelentősebb memória és processzor igénye van. Ajánlott legalább 16GB memória és egy 4 magos processzor, illetve 5 GB szabad hely a lemezen.
- A szerver az első indítás alkalmával letölti a betanított transformer hálózat aktuális verzióját, így az első indításhoz mindenképp szükséges hálózati kapcsolat.
- Operációs rendszerek tekintetében alapvetően nincs megkötés. Szimplán csak működjön az adott rendszeren a Docker Desktop alkalmazás és engedje a virtualizációt valamilyen formában.
- A telepítés során ajánlott a haladó szintű felhasználói ismeret(jegyzék struktúrák, terminál ismerete stb.)
- Az alkalmazást Google Chrome, Firefox, Opera és Microsoft Edge böngészőkön teszteltük, így a futtatása is onnan ajánlott.

Miután meggyőződtünk a fenti követelmények teljesüléséről a következő lépésekkel tudjuk feltelepíteni az alkalmazást:

- Töltsük le és telepítsük fel a Docker Desktop nevű alkalmazást. Az alkalmazás elérhető a `https://www.docker.com/products/docker-desktop` weboldalról Windows, Linux és MacOS rendszerekre is egyaránt. Erre az alkalmazásra a virtualizáció miatt van szükség, ami jelentősen megkönnyíti a telepítést.

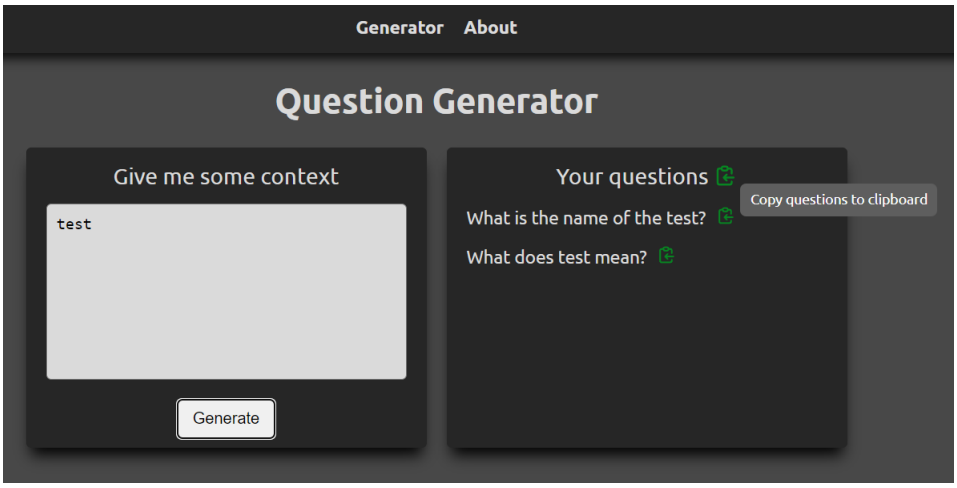
- Lépjünk be terminálon vagy cmd-n keresztül a program mappájába és futtassuk a *docker-compose up* parancsot. Ez a parancs létrehoz egy virtuális konténert külön a szerver és külön a kliens oldali architektúrának.
- Következő lépésben lépjünk be az alkalmazás *qg_app_frontend* mappájába és a *.env.dist* fájl alapján hozzunk létre egy *.env* fájlt.
- A *.env* fájlban a **VITE_BASE_URL** változó a szerver elérhetősége (pl. `http://localhost`), míg a **VITE_ENV** a kliens környezetének beállítására szolgál, ami lehet fejlesztői(dev) vagy éles(prod).
- Ezután elérhetővé válik az alkalmazás kliens oldali része a `http://localhost:3000`-es cím alatt, míg a szerver a `http://localhost:80`-as cím alatt lesz megtalálható.

4.5. Az alkalmazás felhasználói felülete



4.1. ábra. A program felhasználói felülete.

A program nagyon egyszerű felhasználói interfészt használ. Adott 2 oldal: a főoldal és egy az alkalmazás leírását tartalmazó oldal. A főoldalon található maga a kérdésgeneráló komponens. A komponens bal oldalán lehet megadni a szöveges kontextust, ami alapján a kérdések generálódnak, míg a jobb oldalon fognak megjelenni az elkészült kérdések, könnyen másolható formában.



4.2. ábra. Az elkészült kérdések.

4.6. Felhasznált fejlesztői eszközök bemutatása

Mielőtt bemutatnám az alkalmazás részletes működését, szeretném ismertetni a fontosabb felhasznált, elsősorban szerveroldali programkönyvtárakat(API-kat), melyek jelentősen megkönnyítették a fejlesztést. Ezen könyvtárak szinte mindegyike feltelepíthető a Python alapértelmezett csomagkezelőjével, a **pip**-el.

4.6.1. datasets

A **datasets** könyvtár számos hasznos funkciót tartalmaz különböző adatforrások betöltésére, feldolgozására és megosztására. Nagyon sokan használják az NLP, számítógépes látás, adatfeldolgozás területén. Előnye, hogy képes nagy méretű adatforrásokat gyorsan és optimálisan a memóriában tárolni, ahonnan később felhasználhatjuk azt alkalmazásunkban. Legfontosabb metódusa a **load_dataset** metódus, melynek segítségével betölthetjük lokális vagy online formában elérhető adatforrásainkat.

4.6.2. transformers

Ez a könyvtár felelős az előtanított hálózatok kezeléséért. Lehetőség van letölteni különböző betanított neurális hálózatokat, melyeket később tovább taníthatunk(fine-tuning), illetve fel is tölthetünk a Hugging Face Hub nevű platformra, ahonnan futtathatjuk és meg is oszthatjuk másokkal megoldásainkat. Ez a megközelítése a mesterséges intelligencia alapú hálózatoknak úttörő jelentőségű, hiszen nagyon közel áll az emberi agy működéséhez, melynek működési alapelve a folyamatos tanulás és a különböző feladatokra való specializálódás.

Az első fontosabb funkció a könyvtárban a **pipeline** metódus, melynek segítségével

különböző feladatokra specializálódott előtanított hálózatokat használhatunk kódunkban[18]. A metódus meghívásánál egyszerűen csak meg kell neveznünk a kívánt feladatot, választhatunk egy modellt, aminek segítségével fogja megoldani a feladatot, illetve biztosítanunk kell egy paramétert neki, ami a feladattól függően lehet egy kép, hangfájl, szövegrészlet vagy egyéb egyszerűbb értékek is. A metódus eredménye végül egy objektum lesz, mely a megoldásokat tartalmazza és szintén tetszőleges típusú lehet. A végeredmények mellé általában társul valamilyen statisztika is, hogy a modell milyen biztossággal állította elő az adott eredményt.

Egy másik lényeges koncepció a könyvtárban az ún. **tokenizer**-ek, melyek a modellek bemenetének előkészítésében játszanak jelentős szerepet. Működésük során a bemeneti szövegrészeket különböző egységekre bontják és ellátják őket azonosítókkal, majd pedig az elkészült token-eket kódolják[18]. Ezt a folyamatot visszafelé is képesek megcsinálni, így tokenekből képesek dekódolni az elkészült szöveges eredményeket. Az alap funkciókon felül saját kódolást és feldolgozási lépéseket is hozzáadhatunk működésükhöz, ezáltal tudjuk feladatunkhoz szabni őket. Az alap tokenizer osztály a **PreTrainedTokenizerBase**, mely tartalmazza a kódoláshoz, dekódoláshoz, tokenizáláshoz és speciális tokenek definiálásához szükséges funkciókat. Ezt az osztályt kiterjesztve lehet saját tokenizer-t is készíteni.

A modellek tanításához a könyvtár biztosítja a **Trainer** osztályt. Ez az osztály felel az alap tanítási ciklus elkészítéséért. Támogatja az elosztott, párhuzamos tanítást is grafikus processzoron(GPU), illetve tenzor-feldolgozó egységen(TPU). Argumentumként meg kell adnunk a Trainer-nek a modellünket, amit tanítani szeretnénk, a tanító, illetve validáló adathalmazokat, valamint egy **DataCollator** osztályt, ami az adathalmazok csoportokra bontásáért felel, így gyorsítva a tanítást. Ezen kívül megadhatunk egy **TrainingArguments** objektumot neki, ahol beállíthatjuk[18]:

- a különböző hardveren való tanítással kapcsolatos attribútumokat
- a tanulási rátát
- a tanulási ciklusok számát
- hány lépésenként mentse el a hálózat aktuális állapotát
- készítsen-e jelentéseket a hálózat adott pillanataiban a súlyairól és hiba rátájáról
- elmentse-e lokálisan vagy a Hugging Face Hub nevű oldalra az elkészült hálózatot

Az utolsó fontosabb funkciója a könyvtárnak a **modellek**, melyek a már ténylegesen elkészült, betanított neurális hálózatainkat fogják kezelni. A könyvtár által biztosított

PreTrainedModel osztályok tartalmazzák az olyan funkciókat, mint a hálózatok betöltése egy adott lokális vagy online elérhető helyről, a hálózat mentése, illetve a bemeneti tokenek átméretezése a tokenizernek és a hálózatnak megfelelő méretre[18]. Ezen osztályra épül az általam használt **T5ForConditionalGeneration** nevű modell osztály is, mely képes eltárolni és kezelni az előtanított T5 modelleket, illetve kompatibilis a **T5Tokenizer** osztályokkal is, így könnyítve a tanítóhalmazok és a végeredmények feldolgozásán. A T5-höz hasonlóan lehet találni más modell osztályokat is a könyvtárban, például léteznek BERT-hez, GPT-hez és ezek különböző változataihoz készült modellek és tokenizerek is.

4.6.3. torch

A **torch** biztosítja számunkra a hardveres gyorsítást. Segítségével tanítóhalmazunkat eltárolhatjuk ún. tenzorokban, melyek lényegében azonos típusú adatokat tartalmazó többdimenziós mátrixok és ezeket felhasználva tudjuk tanítani neurális hálózat modellünket általános processzoron, grafikus processzoron vagy tenzor-feldolgozó egységen. A **transformers** könyvtár is támogatja a torch-al való hardveres gyorsítás használatát a **DataCollator** osztályokon keresztül, melyek pont a tanítási adatok csoportosításáért felelnek. A könyvtár használatával jelentősen fel tudtuk gyorsítani a tanítási és szöveggenerálási időket alkalmazásunk esetében is.

4.6.4. SQuAD adathalmaz

A **Stanford Question Answering Dataset(SQuAD)[19]** a Stanford Egyetem által készített szabad hozzáférésű szövegértési adathalmaz. Több mint 100 000 kontextus-kérdés-válasz hármasból áll, melyeket Wikipedia cikkek alapján állítottak össze. Az egyes adatrekordokhoz tartozik egy rövidebb szöveges kontextus, 4-5 kérdés, illetve minden kérdéshez néhány helyes válasz. Az egyes válaszoknál az is meg van adva, hogy a kontextusban hol található meg a válasz(ez inkább a kérdés megválaszolási feladatoknál fontos szempont). Alkalmazásunkban ezt az adathalmazt használtuk a kérdésgeneráló T5 alapú neurális hálózat betanítására. Ehhez egy adatbetöltő szkriptet használtunk, mely csatolva lett a dolgozathoz, illetve a `https://huggingface.co/datasets/squad/blob/main/squad.py` URL alatt is elérhető.

4.7. A program részletes működése

4.7.1. A hálózat tanításáért felelős komponens

Mint azt már a tervezési részben említettem a tanításért felelős komponensek hardverigényük miatt szeparált fájlba és környezetbe kerültek a program többi részétől annak érdekében, hogy megfelelő és költséghatékony helyen lehessen őket futtatni. Emiatt a hálózat tanítási részét egy Python alapú *.ipynb* fájlba szerveztük, melyet a Google Colab nevű felhőalapú környezetben tudtunk lefuttatni.

A szükséges könyvtárak importálása után a program kér egy bejelentkezést a Hugging Face Hub nevű platformra, melyre később a már betanított hálózatot fogja menteni, illetve kéri az adathalmazt is, mely esetünkben a csatolt *t5_squad.py* fájl lesz, ami a SQuAD adathalmazt fogja letölteni és megfelelő formátumra alakítani a későbbi feldolgozáshoz.

A következő lépés az adatok előfeldolgozása. Itt már szükséges volt definiálnunk az alap transformer modellünket, ami a *t5-base* lesz, mert az előfeldolgozáshoz szükségünk van egy tokenizer-re, melynek mérete meg kell egyezzen a modell bemeneti token mátrixának méretével, így ezt itt be kell állítanunk. Ezután következhet a nyers adathalmaz tokenekre bontása és kódolása, melyet az alap modellhez tartozó tokenizer segítségével végezhetünk el. A fázis kimenete egy tanítási és egy validációs adathalmaz lesz, melyet a hálózat tanítása során fogunk tudni majd felhasználni.

Miután elkészült a tokenizált és kódolt bemenet, elkezdhetjük a hálózat tanítását. A *torch* könyvtár tenzorai segítségével különböző csoportokra bontjuk a bemenetet egy *DataCollator* osztályban, így lehetővé téve a GPU-n vagy TPU-n való párhuzamosítást. A *Trainer* osztály segítségével és néhány argumentum megadása után végül elindíthatjuk a tanítást. A trainer jelenleg 4-es csoportokban képes az egyes GPU/TPU magokon tanítani a hálózatot, 100 lépésenként jelentést készít, illetve 500 lépésenként menti is a hálózat aktuális állapotát. Természetesen erősebb hardvereken meg lehet próbálni nagyobb csoportokban tanítani a hálózatot, de gyengébb hardvereken gyakran futhatunk ki a memóriából a nagyobb csoportszám miatt.

Utolsó lépésben a szkript feltölti a Hugging Face Hub nevű platformra a hálózatot, ahonnan később a szerverünk letöltheti és használhatja azt.

4.7.2. Szerveroldali komponens

Az alkalmazás szerveroldali nyelve a tanító komponenshez hasonlóan Python, mivel így sokkal egyszerűbben tudtuk kezelni a neurális hálózatot. A szerver Django keretrendszer segítségével lett elkészítve, mely biztosítja számunkra a könnyű és gyors futtatást

és az eszköztárakat. A szerver konfigurációját tekintve nyílt, így lényegében bármely alkalmazás elérheti az interneten keresztül. Jelenleg adatbázist nem tartalmaz a szerver, így felhasználói és egyéb alkalmazás adatok nem kerülnek tárolásra a futás során.

Szerveroldalon az egyetlen végpont, amit implementálnunk kellett az a *api/generate-questions* nevű útvonal. Ez felel majd a megadott kontextushoz generált kérdések elkészítéséért. A *transformers* könyvtárt itt is telepíteni kellett, hiszen szükségünk van a tokenizer-ünkre és a modellünkre a kérdések generálásához. A tokenizer és a modell definiálása után a megadott szöveges kontextust a tokenizer segítségével kódoljuk és átadjuk a modellünknek, hogy generáljon ez alapján kérdéseket. A generáláshoz, akár csak a modell tanításához számos paramétert adhatunk meg. Változtathatjuk a maximálisan kigenerálható kérdések mennyiségét, a kérdések hosszát, az egyes szavak maximális ismétlődését, illetve a dekódolási algoritmust is beállíthatjuk, ami lehet mohó keresés, sugár keresés, top-k mintavétel vagy top-p mintavétel is.

A generált és dekódolt kimeneti kérdéseket végül szöveges tömb formájában adja vissza a szerver, JSON formátumban, amit aztán a kliens fel tud dolgozni.

4.7.3. Kliensoldali komponens

A kliensoldal TypeScript-ben íródott és a Vue.js nevű frontend keretrendszert használtuk az elkészítéséhez, ami biztosítja a HTML, CSS, és JavaScript kódok összecsomagolását, optimalizálást, illetve a kódolást is jelentősen megkönnyíti. A kliensoldali működés oldalakra és komponensekre lett bontva. Két oldalt használ az alkalmazás: a főoldalt, ahol a kérdésgenerátor van, valamint egy leírás oldalt, ahol az alkalmazás használata és bemutatása van részletezve.

Komponensek közül a *QuestionGenerator* a legfontosabb, amely magát a kérdésgenerálás kliensoldali részét tartalmazza. Itt adott egy szövegdoboz, amibe a felhasználó beírhatja a szöveges kontextust, ami alapján szeretne kérdéseket generálni. Ezután a komponens kapcsolatot létesít a szerverrel és sikeres szerverválasz esetén megjeleníti az elkészült kérdéseket. A kérdések ezután egy segédfüggvény segítségével kimásolhatók és szabadon beilleszthetők bármilyen dokumentumba.

5. fejezet

Tesztelés

Ebben a fejezetben kerülnek bemutatásra az alkalmazáshoz készült automatikus, illetve manuális tesztek.

5.1. Automatikus tesztek

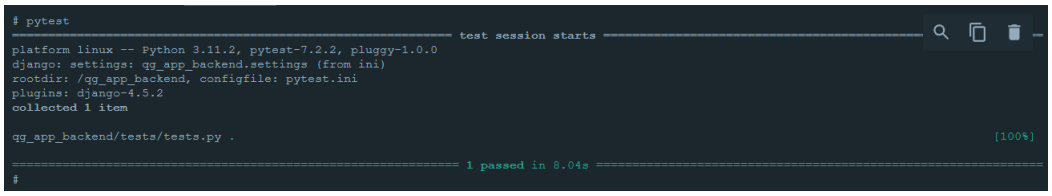
Modern webalkalmazás lévén esetünkben is elengedhetetlen az automatikus tesztelés. Minden egyes funkcióhoz készültek egységtesztek, melyek biztosítják a megfelelő működésüket.

Szerveroldalon egyetlen fontos végpont van, amit tesztelni kellett, az pedig a */api/generate-questions* útvonal.

```python
import json

def testGenerateQuestions(client):
    headerInfo = {'content-type': 'application/json' }
    payload = {'context': 'Test.'}

    response = client.post("/api/generate-questions", headers=headerInfo, data=payload)

    assert response.status_code == 200
    assert len(response.data) > 0
    assert type(response.data[0]) == str
```

5.1. ábra. A szerveroldali végpont egységtesztje.

A **qg_app_backend** konténerbe terminállal belépve a *pytest* parancs futtatásával lehet elindítani a szerveroldali teszteket. Ekkor minden **test** szót tartalmazó fájlt tesztnek érzékel és lefuttat a tesztkörnyezet.



5.2. ábra. A szerveroldali tesztek eredménye.

Kliens oldalon minden komponenshez készült egységteszt, melyek az adott komponenssel egy mappába kerültek elhelyezésre. Készült továbbá néhány integrációs teszt is az egyes oldalakhoz, illetve az alkalmazás egészéhez.

```
import { shallowMount } from "@vue/test-utils";
import { expect, test } from "vitest";
import Question from "@/components/question_generator/Question.vue";

test("mount component", () => {
  const mockText = "test";

  const wrapper = shallowMount(Question, options: {
    props: {
      text: mockText,
    },
  });

  expect(wrapper.html()).toMatchSnapshot();
  expect(wrapper.text()).toContain(mockText);
});
```

5.3. ábra. Question nevű komponens egységtesztje.

```
import { mount } from "@vue/test-utils";
import { expect, test } from "vitest";
import HomePage from "@/pages/HomePage.vue";

test("Home page renders", () => {
  const wrapper = mount(HomePage);

  expect(wrapper.text()).toContain("Question Generator");
  expect(wrapper.text()).toContain("Give me some context");
  expect(wrapper.text()).toContain("Your questions");
});
```

5.4. ábra. A főoldal integrációs tesztje.

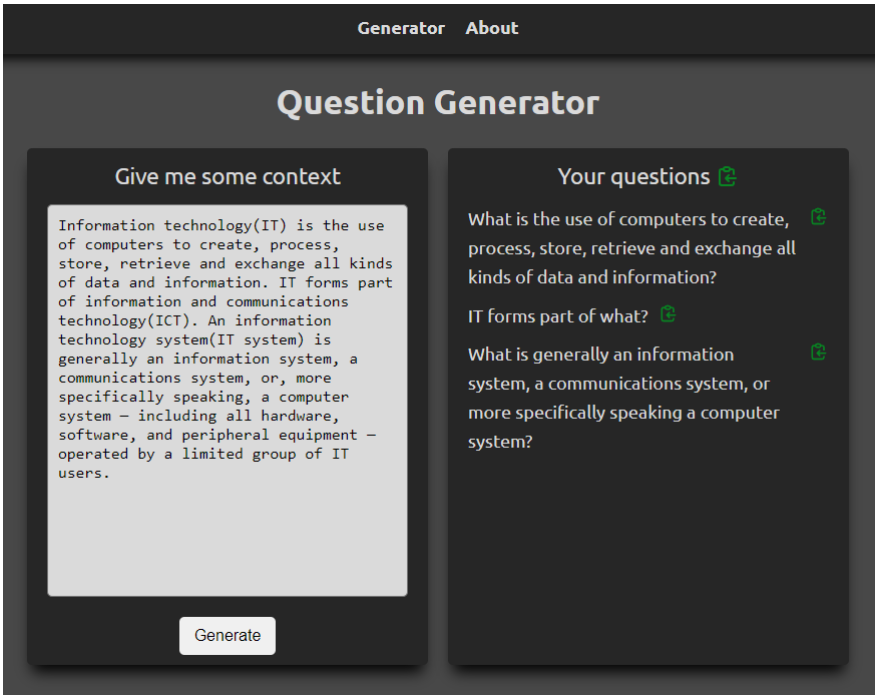
A kliensoldai egységtesztek futtatásához a **qg_app_frontend** konténerbe kell belépnünk és el kell indítanunk a *npm run test* parancsot. Ekkor szintén minden kliensoldali **test** szót tartalmazó fájlban lefutnak a tesztek.

```
/qg_app_frontend # npm run test

> qg_app_frontend@1.0.0 test
> vitest --environment jsdom


 DEV  v0.29.2 /qg_app_frontend

 ✓ src/components/tooltip/Tooltip.unit.test.ts (1)
 ✓ src/components/question_generator/Question.unit.test.js (1)
 ✓ src/utils/useClipboard.unit.test.ts (1)
 ✓ src/components/question_generator/Question.unit.test.ts (1)
 ✓ src/pages/HomePage.integ.test.ts (1)
 ✓ src/pages/AboutPage.integ.test.ts (1)
 ✓ src/components/question_generator/QuestionGenerator.unit.test.ts (1)

 Test Files  7 passed (7)
      Tests  7 passed (7)
   Start at  12:08:06
   Duration  40.73s (transform 852ms, setup 0ms, collect 11.62s, tests 303ms)


 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

5.5. ábra. A kliensoldali tesztek eredménye.

5.2. Manuális tesztek

Mivel alkalmazásunk egy NLP feladat megoldására íródott, így elengedhetetlen, hogy kézzel is teszteljük. Ez esetünkben annyit jelentett, hogy különböző kontextusokat adtunk meg az alkalmazásnak és figyeltük, hogy milyen kérdéseket generált. Egy kérdés akkor számít elfogadottnak, ha értelmes, van köze a kontextushoz és nyelvtanilag is helyes.



5.6. ábra. Az alkalmazás egy manuális tesztje.

A kezdeti tesztek elsősorban az alap működés ellenőrzésére szolgáltak, vagyis hogy képes-e az alkalmazás szavakból, mondatokból vagy hosszabb szövegekből kérdéseket generálni.

- **Kontextus:** "one"
- **Kérdések:**
  - How many questions does one have to answer in order to get a good answer?

- **Kontextus:** "hungarian"
- **Kérdések:**
  - What is the nationality of the hungarian population?
  - What country is the capital of Hungary?

- **Kontextus:** "Hungary is a country."
- **Kérdések:**
  - What is Hungary?
  - What is a country?

- **Kontextus:** "Kate made an apple pie."
- **Kérdések:**
  - Kate made what kind of pie?

- **Kontextus:** "Information technology(IT) is the use of computers to create, process, store, retrieve and exchange all kinds of data and information. IT forms part of information and communications technology(ICT). An information technology system(IT system) is generally an information system, a communications system, or, more specifically speaking, a computer system — including all hardware, software, and peripheral equipment — operated by a limited group of IT users."
- **Kérdések:**
  - What is the use of computers to create, process, store, retrieve and exchange all kinds of data and information?
  - IT forms part of what?

  - What is generally an information system, a communications system, or more specifically speaking a computer system?

- **Kontextus:** "A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. It is used primarily in the fields of natural language processing (NLP) and computer vision (CV). Like recurrent neural networks (RNNs), transformers are designed to process sequential input data, such as natural language, with applications towards tasks such as translation and text summarization. However, unlike RNNs, transformers process the entire input all at once. The attention mechanism provides context for any position in the input sequence. For example, if the input data is a natural language sentence, the transformer does not have to process one word at a time. This allows for more parallelization than RNNs and therefore reduces training times."

- **Kérdések:**

  - What is a deep learning model that adopts the mechanism of self-attention?
  - What is used primarily in the fields of natural language processing (NLP) and computer vision (CV)?
  - Like recurrent neural networks, transformers are designed to process what input data?

Ezt követően elkezdtünk az alkalmazásnak különböző témakörökből kontextusokat megadni, hiszen célunk, hogy minél színesebb témaválasztékból tudjon a program kérdéseket generálni.

- **Témakör:** Fizika

- **Kontextus:** "The Schrödinger equation is a linear partial differential equation that governs the wave function of a quantum-mechanical system. It is a key result in quantum mechanics, and its discovery was a significant landmark in the development of the subject. The equation is named after Erwin Schrödinger, who postulated the equation in 1925, and published it in 1926, forming the basis for the work that resulted in his Nobel Prize in Physics in 1933. Conceptually, the Schrödinger equation is the quantum counterpart of Newton's second law in classical mechanics. Given a set of known initial conditions, Newton's second law makes a mathematical prediction as to what path a given physical system will take over time. The Schrödinger equation gives the evolution over time of a wave function, the quantum-mechanical characterization of an isolated physical system.

The equation can be derived from the fact that the time-evolution operator must be unitary, and must therefore be generated by the exponential of a self-adjoint operator, which is the quantum Hamiltonian. The Schrödinger equation is not the only way to study quantum mechanical systems and make predictions. The other formulations of quantum mechanics include matrix mechanics, introduced by Werner Heisenberg, and the path integral formulation, developed chiefly by Richard Feynman. Paul Dirac incorporated matrix mechanics and the Schrödinger equation into a single formulation. When these approaches are compared, the use of the Schrödinger equation is sometimes called "wave mechanics"."

- **Kérdések:**
  - What is a linear partial differential equation that governs the wave function of a quantum-mechanical system?
  - Who is the Schrödinger equation named after?
  - What is the quantum counterpart of Newton's second law in classical mechanics?
  - Which formulation of quantum mechanics was developed by Richard Feynman?

- **Témakör:** Történelem
- **Kontextus:** "Hungary in its modern (post-1946) borders roughly corresponds to the Great Hungarian Plain (the Pannonian Basin). During the Iron Age, it was located at the crossroads between the cultural spheres of the Celtic tribes (such as the Scordisci, Boii and Veneti), Dalmatian tribes (such as the Dalmatae, Histri and Liburni) and the Germanic tribes (such as the Lugii and Marcomanni). The name "Pannonian" comes from Pannonia, a province of the Roman Empire. Only the western part of the territory (the so-called Transdanubia) of modern Hungary formed part of Pannonia. The Roman control collapsed with the Hunnic invasions of 370–410, and Pannonia was part of the Ostrogothic Kingdom during the late 5th to mid 6th century, succeeded by the Avar Khaganate (6th to 9th centuries). The Magyar invasion took place during the 9th century. The Magyars were Christianized at the end of the 10th century, and the Christian Kingdom of Hungary was established in 1000 under King Saint Stephen, ruled by the Árpád dynasty for the following three centuries. In the high medieval period, the kingdom expanded to the Adriatic coast and entered a personal union with Croatia during the reign of King Coloman in 1102. In 1241 during the reign of King Béla IV, Hungary was invaded by the Mongols under Batu Khan. The outnumbered

Hungarians were decisively defeated at the Battle of Mohi by the Mongol army. In this invasion more than 500,000 Hungarian people were massacred and the whole kingdom reduced to ashes. The paternal lineage of the ruling Árpád dynasty came to end in 1301, and all of the subsequent kings of Hungary (with the exception of King Matthias Corvinus) were cognatic descendants of the Árpád dynasty. Hungary bore the brunt of the Ottoman wars in Europe during the 15th century. The peak of this struggle took place during the reign of Matthias Corvinus (r. 1458–1490). The Ottoman–Hungarian wars concluded in significant loss of territory and the partition of the kingdom after the Battle of Mohács of 1526."

- **Kérdések:**
  - What is the Pannonian Basin?
  - Who invaded Hungary in 1241?
  - How many people were massacred in the Battle of Mohi?
  - When did the Ottoman-Hungarian wars end?

- **Témakör:** Biológia
- **Kontextus:** "A virus is a submicroscopic infectious agent that replicates only inside the living cells of an organism. Viruses infect all life forms, from animals and plants to microorganisms, including bacteria and archaea. Since Dmitri Ivanovsky's 1892 article describing a non-bacterial pathogen infecting tobacco plants and the discovery of the tobacco mosaic virus by Martinus Beijerinck in 1898, more than 9,000 of the millions of virus species have been described in detail. Viruses are found in almost every ecosystem on Earth and are the most numerous type of biological entity. The study of viruses is known as virology, a subspeciality of microbiology. When infected, a host cell is often forced to rapidly produce thousands of copies of the original virus. When not inside an infected cell or in the process of infecting a cell, viruses exist in the form of independent viral particles, or virions, consisting of the genetic material, i.e., long molecules of DNA or RNA that encode the structure of the proteins by which the virus acts; a protein coat, the capsid, which surrounds and protects the genetic material; and in some cases an outside envelope of lipids. The shapes of these virus particles range from simple helical and icosahedral forms to more complex structures. Most virus species have virions too small to be seen with an optical microscope and are one-hundredth the size of most bacteria. The origins of viruses in the evolutionary history of life are unclear: some may have evolved from plasmids—pieces of DNA that can move between cells—while others may have evolved from bacteria. In evolution,

viruses are an important means of horizontal gene transfer, which increases genetic diversity in a way analogous to sexual reproduction. Viruses are considered by some biologists to be a life form, because they carry genetic material, reproduce, and evolve through natural selection, although they lack the key characteristics, such as cell structure, that are generally considered necessary criteria for defining life. Because they possess some but not all such qualities, viruses have been described as "organisms at the edge of life" and as replicators."

- **Kérdések:**
  - What is a submicroscopic infectious agent that replicates only inside the living cells of an organism?
  - How many of the millions of virus species have been described in detail since Dmitri Ivanovsky's 1892 article?
  - What is the study of viruses known as?

- **Témakör:** Informatika
- **Kontextus:** "In computing, a server is a piece of computer hardware or software (computer program) that provides functionality for other programs or devices, called "clients." This architecture is called the client–server model. Servers can provide various functionalities, often called "services," such as sharing data or resources among multiple clients or performing computations for a client. A single server can serve multiple clients, and a single client can use multiple servers. A client process may run on the same device or may connect over a network to a server on a different device. Typical servers are database servers, file servers, mail servers, print servers, web servers, game servers, and application servers. Client–server systems are usually most frequently implemented by (and often identified with) the request–response model: a client sends a request to the server, which performs some action and sends a response back to the client, typically with a result or acknowledgment. Designating a computer as "server-class hardware" implies that it is specialized for running servers on it. This often implies that it is more powerful and reliable than standard personal computers, but alternatively, large computing clusters may be composed of many relatively simple, replaceable server components."
- **Kérdések:**
  - What is a piece of computer hardware or software that provides functionality for other programs or devices called?

  - What architecture is called the client-server model?
  - Servers can provide various functionalities, often called what?
  - A single server can serve multiple clients, and a single client can use multiple servers?

- **Témakör:** Földrajz
- **Kontextus:** "Earth is the third planet from the Sun and the only place known in the universe where life has originated and found habitability. While Earth may not contain the largest volumes of water in the Solar System, only Earth sustains liquid surface water, extending over 70.8% of the Earth with its ocean, making Earth an ocean world. Earth's polar regions currently retain most of all other water with large sheets of ice covering ocean and land, dwarfing Earth's groundwater, lakes, rivers and atmospheric water. Land, consisting of continents and islands, extends over 29.2% of the Earth and is widely covered by vegetation. Below Earth's surface material lies Earth's crust consisting of several slowly moving tectonic plates, which interact to produce mountain ranges, volcanoes, and earthquakes. Earth's liquid outer core generates a magnetic field that shapes the magnetosphere of Earth, largely deflecting destructive solar winds and cosmic radiation."
- **Kérdések:**
  - What planet is the third from the Sun?
  - What is the only place known in the universe where life has originated and found habitability?
  - Earth's liquid outer core generates a magnetic field that shapes the magnetosphere of what?

5.3. Összehasonlítás a ChatGPT-vel

Mivel mostanság nagyon elterjedtek a különböző általános NLP feladatokat megoldó szöveggenerátorok, így elengedhetetlen volt, hogy alkalmazásunkat összehasonlítsuk egy másik, hasonló programmal. A ChatGPT-re esett a választás, mivel jelenleg ez az egyik legnépszerűbb és legjobb teljesítményt elért program a piacon.

- **Kontextus:** "Thomas ate an apple."
- **Alkalmazásunk kérdései:**

  - What did Thomas eat?

- **ChatGPT kérdései:**

  - What did Thomas eat?

- **Kontextus:** "Claire went to the store and bought some vegetables, fish, butter and shampoo."

- **Alkalmazásunk kérdései:**

  - Claire went to the store and bought what?
  - What did Claire buy at the store?

- **ChatGPT kérdései:**

  - What items did Claire purchase when she went to the store?

- **Kontextus:** "One day, a fox became very hungry as he went to search for some food. He searched high and low, but couldn't find something that he could eat. Finally, as his stomach rumbled, he stumbled upon a farmer's wall. At the top of the wall, he saw the biggest, juiciest grapes he'd ever seen. They had a rich, purple color, telling the fox they were ready to be eaten. To reach the grapes, the fox had to jump high in the air. As he jumped, he opened his mouth to catch the grapes, but he missed. The fox tried again but missed yet again. He tried a few more times but kept failing. Finally, the fox decided it was time to give up and go home. While he walked away, he muttered, "I'm sure the grapes were sour anyway."

- **Alkalmazásunk kérdései:**

  - What did a fox do when he became hungry?
  - What color did the grapes on the wall tell the fox they were ready to be eaten?
  - The fox had to jump high in the air to reach what?

- **ChatGPT kérdései:**

  - What did the fox do when he became very hungry?
  - Where did the fox search for food?
  - What did the fox find when he stumbled upon a farmer's wall?
  - How did the fox feel when he saw the grapes?

  - What did the fox have to do to reach the grapes?

- **Kontextus:** "The London Zoo, which was opened in 1828, was initially known as the "Gardens and Menagerie of the Zoological Society of London", and it described itself as a menagerie or "zoological forest". The abbreviation "zoo" first appeared in print in the United Kingdom around 1847, when it was used for the Clifton Zoo, but it was not until some 20 years later that the shortened form became popular in the rhyming song "Walking in the Zoo" by music-hall artist Alfred Vance. The term "zoological park" was used for more expansive facilities in Halifax, Nova Scotia, Washington, D.C., and the Bronx in New York, which opened in 1847, 1891 and 1899 respectively. Relatively new terms for zoos in the late 20th century are "conservation park" or "bio park". Adopting a new name is a strategy used by some zoo professionals to distance their institutions from the stereotypical and nowadays criticized zoo concept of the 19th century. The term "bio park" was first coined and developed by the National Zoo in Washington D.C. in the late 1980s. In 1993, the New York Zoological Society changed its name to the Wildlife Conservation Society and re branded the zoos under its jurisdiction as "wildlife conservation parks"."

- **Alkalmazásunk kérdései:**
  - When was the London Zoo opened?
  - When did the abbreviation "zoo" first appear in print in the United Kingdom?
  - What did Alfred Vance's song "Walking in the Zoo" become popular in?
  - In what year did the New York Zoological Society change its name to?

- **ChatGPT kérdései:**
  - When was the London Zoo opened and what was its original name?
  - How did the abbreviation "zoo" come into use in the United Kingdom?
  - Who popularized the term "zoo" in a rhyming song and when was it released?
  - What are some examples of facilities that were referred to as "zoological parks" in the 19th century?
  - What are some of the relatively new terms used for zoos in the late 20th century?
  - Why do some zoo professionals adopt new names for their institutions?

  - Which zoo first coined the term "bio park" and when was it developed?
  - In what year did the New York Zoological Society change its name and rebrand its zoos as "wildlife conservation parks"?

- **Kontextus:** "In computing, a database is an organized collection of data stored and accessed electronically. Small databases can be stored on a file system, while large databases are hosted on computer clusters or cloud storage. The design of databases spans formal techniques and practical considerations, including data modeling, efficient data representation and storage, query languages, security and privacy of sensitive data, and distributed computing issues, including supporting concurrent access and fault tolerance. A database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a database system. Often the term "database" is also used loosely to refer to any of the DBMS, the database system or an application associated with the database. Computer scientists may classify database management systems according to the database models that they support. Relational databases became dominant in the 1980s. These model data as rows and columns in a series of tables, and the vast majority use SQL for writing and querying data. In the 2000s, non-relational databases became popular, collectively referred to as NoSQL, because they use different query languages."

- **Alkalmazásunk kérdései:**
  - What is an organized collection of data stored and accessed electronically?
  - Where are large databases hosted?
  - What is the software that interacts with end users, applications, and the database itself to capture and analyze the data?
  - When did relational databases become dominant?

- **ChatGPT kérdései:**
  - What is a database in computing?
  - How are small databases stored compared to large databases?
  - What are some practical considerations in the design of databases?
  - What is a database management system (DBMS)?
  - What is the role of DBMS software in capturing and analyzing data?

6. fejezet

Összegzés

Hasonló szerepe van, mint a bevezetésnek. Itt már múltidőben lehet beszélni. A szerző saját meglátása szerint kell összegezni és értékelni a dolgozat fontosabb eredményeit. Meg lehet benne említeni, hogy mi az ami jobban, mi az ami kevésbé jobban sikerült a tervezettnél. El lehet benne mondani, hogy milyen további tervek, fejlesztési lehetőségek vannak még a témával kapcsolatban.

7. fejezet

Summary

The content of the previous chapter in english.

Irodalomjegyzék

[1] Johri, Prashant & Khatri, Sunil Kumar & Al-Taani, Ahmad & Sabharwal, Munish & Suvanov, Shakhzod & Chauhan, Avneesh. (2021). Natural Language Processing: History, Evolution, Application, and Future Work. 10.1007/978-981-15-9712-1_31.

[2] John Hutchins. (2006). The first public demonstration of machine translation : the Georgetown-IBM system , 7 th January 1954

[3] Mounin, Georges. (1961). Chomsky, Noam: Syntactic Structures. Babel. 7.10.1075/babel.7.1.13mou.

[4] Woods, William A. (1970). Transition Network Grammars for Natural Language Analysis

[5] Bánki Dezső & Bognár László & Garai Zsolt & Forrai Gábor & Margitay Tihamér & Máté András & Mekis Péter & Tanács János & Zemplén Gábor. (2006). Esszéírás és informális logika

[6] Le, NT., Kojiri, T., Pinkwart, N. (2014). Automatic Question Generation for Educational Applications – The State of Art.

[7] John Harmon Wolfe. (1976). Automatic question generation from text - an aid to independent study.

[8] Chen, W., Aist, G., & Mostow, J. (2009). Generating Questions Automatically from Informational Text

[9] Jouault, C. and Seta, Kazuhisa. (2013). Building a semantic open learning space with adaptive question generation support

[10] Recurrent neural network unfold, `https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg`

[11] Vaswani, Ashish and Shazeer, Noam and Parmar, Niki and Uszkoreit, Jakob and Jones, Llion and Gomez, Aidan and Kaiser, Lukasz and Polosukhin, Illia. (2017). Attention Is All You Need.

[12] Bahdanau, Dzmitry and Cho, Kyunghyun and Bengio, Y.. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. ArXiv. 1409.

[13] Tassopoulou, Vasiliki. (2019). An Exploration of Deep Learning Architectures for Handwritten Text Recognition. 10.13140/RG.2.2.34041.62565.

[14] Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[15] Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html

[16] Tom B. Brown and Benjamin Mann and Nick Ryder and Melanie Subbiah and Jared Kaplan and Prafulla Dhariwal and Arvind Neelakantan and Pranav Shyam and Girish Sastry and Amanda Askell and Sandhini Agarwal and Ariel Herbert-Voss and Gretchen Krueger and Tom Henighan and Rewon Child and Aditya Ramesh and Daniel M. Ziegler and Jeffrey Wu and Clemens Winter and Christopher Hesse and Mark Chen and Eric Sigler and Mateusz Litwin and Scott Gray and Benjamin Chess and Jack Clark and Christopher Berner and Sam McCandlish and Alec Radford and Ilya Sutskever and Dario Amodei. (2020). Language Models are Few-Shot Learners

[17] Sewon Min and Xinxi Lyu and Ari Holtzman and Mikel Artetxe and Mike Lewis and Hannaneh Hajishirzi and Luke Zettlemoyer. (2022). Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?

[18] Hugging Face Documentations https://huggingface.co/docs/

[19] Pranav Rajpurkar and Jian Zhang and Konstantin Lopyrev and Percy Liang. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text

Az internetes források utolsó ellenőrzése: 2023.04.16

CD melléklet tartalma

A dolgozathoz mellékelt lemezen egy `Dolgozat` nevű jegyzékben a következő fájlok találhatóak.

- A dolgozat LaTeX forráskódja.
- A dolgozat PDF formátumban (`dolgozat.pdf`).
- A magyar és angol nyelvű összefoglaló LaTeX és PDF formátumban (`osszegzes.tex`, `osszegzes.pdf`, `summary.tex`, `summary.pdf`).

A `Program` nevű jegyzékben található a dolgozathoz elkészített program forráskódja és futtatható változata.

- *Feladattól, technológiától függően ez változhat.*
- *Konkretizálni kell, hogy pontosan mit tartalmaz a jegyzék!*