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 kultúrális igényű szövegeket készíteni? Tudnak-e kérdéseket megfogalmazni vagy éppen válaszokat? Ezekre a kérdésekre fogom keresni a választ diplomamunkámban. Bemutatom továbbá egy példaprogramon keresztül a jelenleg rendelkezésünkre álló fejlesztői környezetet és könyvtárakat, valamint a szöveggenerálás során jelenleg használt legfejlettebb algoritmusokat is.

2. fejezet

Természetes nyelvfeldolgozás

2.1. Történeti áttekintés [1]

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.

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.

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 vannak-e ragok az adott nyelvben, milyen igék, tárgyak vannak, illetve ismernie kell a különböző mondathatároló karakterek jelentéseit. 1957-ben **Noam Chomsky** könyvében 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 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.

1960-ban **Terry Winograd** elkészítette **SHRDLU** nevű programját, 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. 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. 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 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 szinte 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:

- 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

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

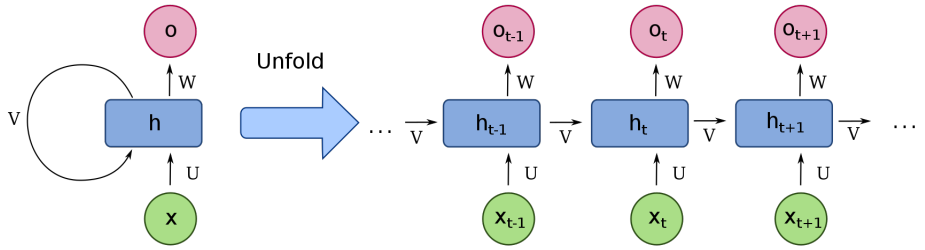
Mindezen problémák miatt logikus lépés volt megvizsgálni az emberi agyat és előállni egy olyan koncepcióval, mely képes modellezni az agy mintafelismerési képességét és ezáltal új távlatokat nyitni a nyelvfeldolgozás területén. Ezek voltak a **neurális hálózatok**.

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 [2].

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.

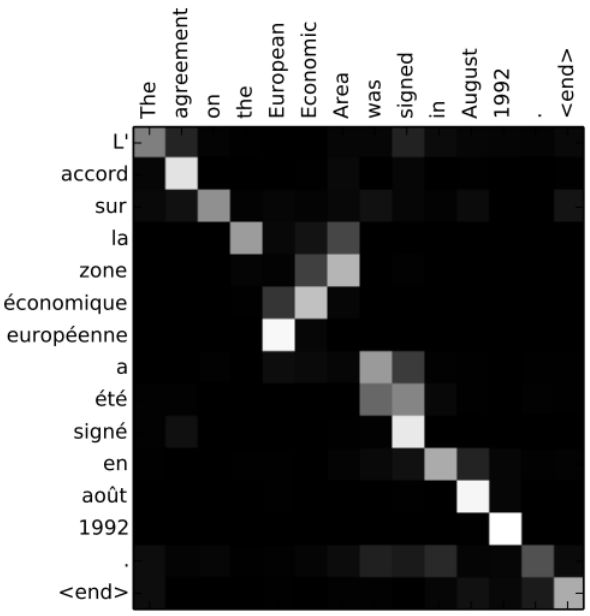
Az átalakítókat a Google és a Torontói Egyetem szakemberei fejlesztették ki 2017-ben és meg is jelentettek egy cikket "Attention Is All You Need"[3] címmel, amiben részletezték ezen átalakítók működését. Talán a legnagyobb újítás a párhuzamosíthatóság területén jelent meg, mivel ezeket a hálózatokat 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 [4].

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

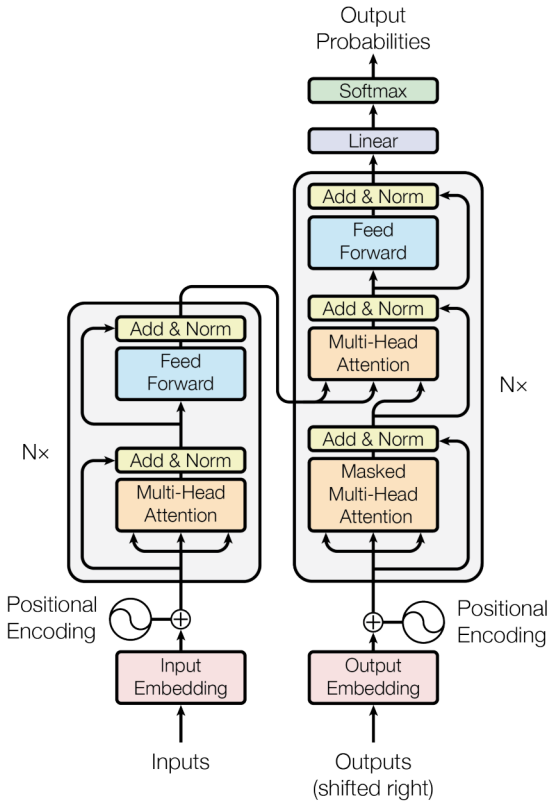
álló neurális hálózat felépítsen egy belső reprezentációt az adott nyelvről. Ez a belső működés leginkább a különböző képfelismerési hálózatok(CNN) rétegeihez hasonló, ahol az egyes régetek képesek felismerni éleket, alakzatok és egyéb magasabb szintű, komplexebb struktúrákat, mint emberek, állatok vagy 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. 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



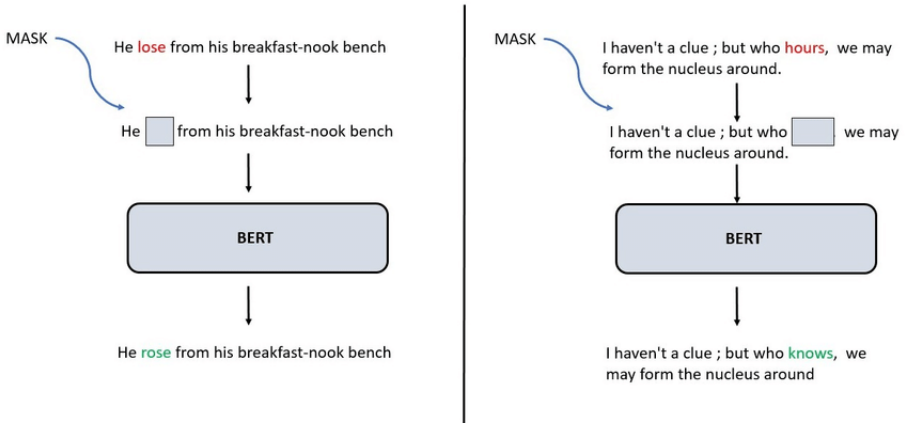
3.3. ábra. Az átalakító modell architektúrája [3].

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)

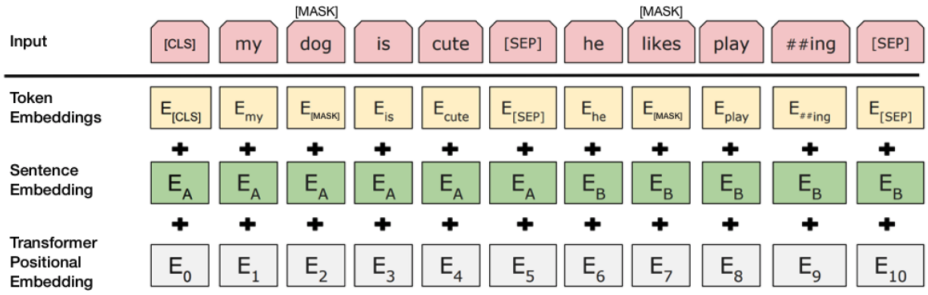
A **Masked-Language Modeling** az adott szöveg, pontosabban a szöveg szavainak mélyebb megértését célozza. A BERT hálózat tanítása során az egyes mondatok szavainak kb. 15%-át kicserélik [MASK] tokenekre. Ezt követően a modell megpróbálja kitalálni ezeket a maszkolt szavakat a körülötte lévő nem maszkolt 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.

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.



3.4. ábra. Maszkolt szavak beillesztése a BERT működése során [5].

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 [6].

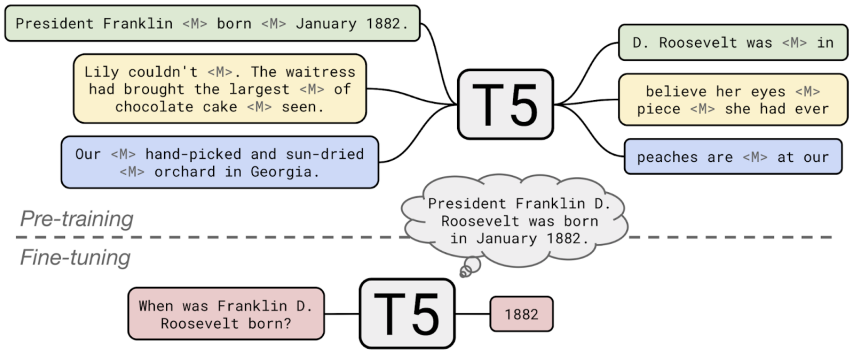
3.4. A T5 modell

A BERT jelentős sikerei után elkezdődtek a kutatások a modell felhasználók számára való 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 szöveg 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 ta-

nított 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 szó lesz, hanem egy generált szöveg, tetszőleges mérettel.



3.6. ábra. T5 modell a tanítás előtt és után [7].

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.

3.5. A GPT-3 modell

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 én végül a neurális hálózat alapú megoldás mellett döntöttem, 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 alkotnia. 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 hálózat létezik, melyekkel megoldható a feladat, azonban a legtöbbjük vagy már elavult, vagy fizetős az elérése, így végül a modernebb és nagyobb tudású, ingyenesen elérhető T5 nevű modell mellett döntöttem.

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.

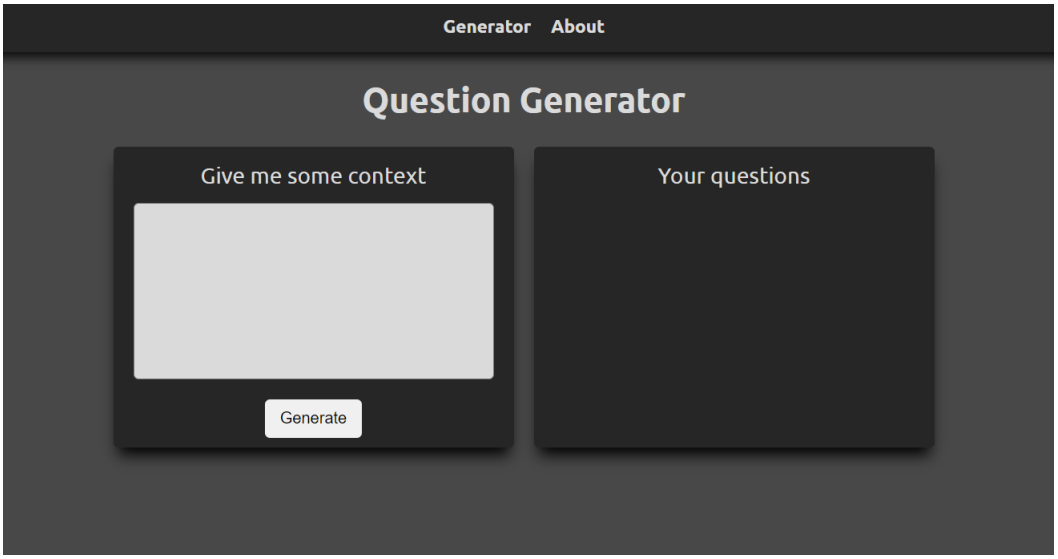
Az alkalmazás lokális elindításához elengedhetetlen néhány alapkövetelmény, melyeknek rendszerünknek meg kell felelnie:

- 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:

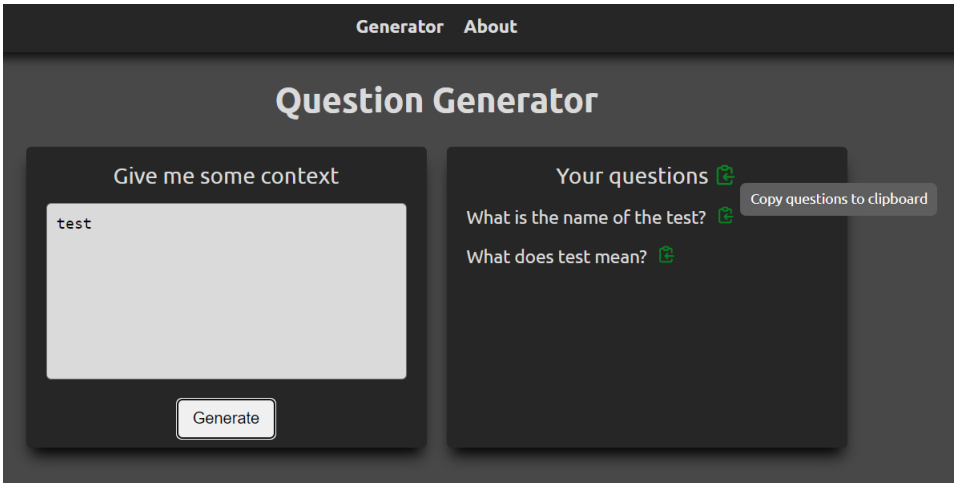
- Töltsük le és telepítsük fel a Docker Desktop nevű alkalmazást. Az alkalmazás elérhető a `https://www.docker.com/products/docker-desktop` weboldalról Windows, Linux és MacOS rendszerekre is egyaránt. Erre az alkalmazásra a virtualizáció miatt van szükség, ami jelentősen megkönnyíti a telepítést.
-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

Mielőtt bemutatnám az alkalmazás részletes működését, szeretném ismertetni a felhasznált szerveroldali programkönyvtárakat(API-kat), melyek jelentősen megkönnyítették a fejlesztést.

4.7. A program részletes működése

5. fejezet

Tesztelés

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

5.1. Automatikus tesztek

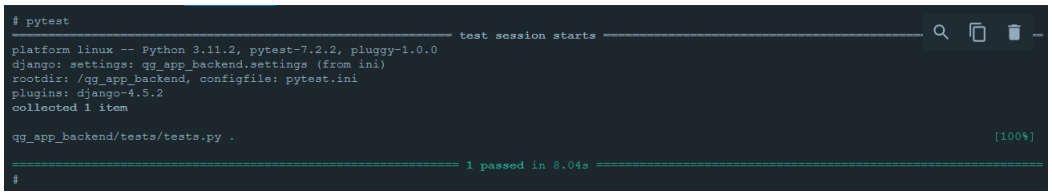
Modern webalkalmazás lévén esetünkben is elengedhetetlen az automatikus tesztelés. Minden egyes funkcióhoz készültek egységtesztek, melyek biztosítják a megfelelő működésüket.

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.

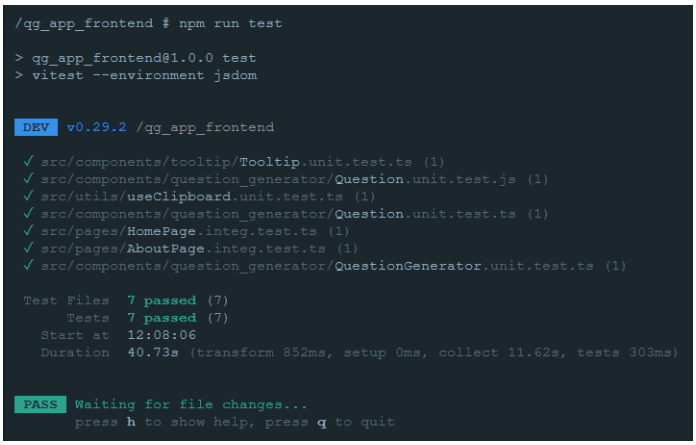
```
import { shallowMount } from "@vue/test-utils";
import { expect, test } from "vitest";
import Question from "@/components/question_generator/Question.vue";

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.



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.

A kezdeti tesztek elsősorban az alap működés ellenőrzésére szolgáltak, vagyis hogy képes-e az alkalmazás szavakból, mondatokból vagy hosszabb szövegből 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?

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

[3] 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.

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

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

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

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

Az internetes források utolsó ellenőrzése: 2022.08.23

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!*