LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

DATORA ĢENERĒTA DZEJA

MAĢISTRA DARBS

Autors: **Laura Jozefa**

Stud. apl. nr. lj13026

Darba vadītājs: Dr. sc. comp. Kaspars Balodis

RĪGA 2022

ANOTĀCIJA

Pēdējos gados ir bijuši vairāki mēģinājumi ar datora palīdzību automātiski ģenerēt arī ko radošu – vizuālās mākslas radīšana, radoša teksta ģenerēšana vai pat mūzikas skaņdarbu komponēšana – šīs ir tikai dažas no jomām, ar kurām darbojas mākslīgais intelekts. Dabiskās valodas ģenerēšana ir visai daudzsološa mākslīgā intelekta un datorlingvistikas apakšnozare, kuras galvenais mērķis ir izstrādāt datorprogrammas, kas spēj radīt cilvēkiem uztveramu tekstu. Starp automātiski ģenerētajiem teksta veidiem var atrast pat dažādas laika prognozes, biogrāfijas, kā arī tekstus, kas ietver sevī radošumu – dažādus stāstus, jokus vai dzeju. Automātiskā dzejas ģenerēšana ir šķietami sarežģīts uzdevums un ļoti interesants mākslīgā intelekta pētījumu temats.

Maģistra darba mērķis ir parādīt, ka valodas modeļi patiešām spēj automātiski ģenerēt tekstus, ko var uzskatīt par poētiskiem, kā arī pielietot valodas modeli, lai ģenerētu dzeju latviešu valodā. Darba gaitā veikts teorijas apskats un datora ģenerētas dzejas vēstures apskats. Apskatīti un salīdzināti dažādi datora ģenerēta teksta veidošanas algoritmi. Ir izvērtēts, kurš modelis ir atbilstošāks individuālai izmantošanai ar ierobežotiem skaitļošanas resursiem dzejas ģenerēšanai un veikta tā izpēte. Balstoties uz izpētīto literatūru un iegūtajām zināšanām, izvēlēts un papildināts valodas modelis, veicot dzejas ģenerēšanu latviešu valodā.

Atslēgvārdi: dabiskās valodas apstrāde, *Transformer* modelis, GPT-2 modelis, datora ģenerēta dzeja.

ABSTRACT

In recent years, there have been several attempts to automatically generate something creative with the help of a computer - the creation of visual art, the generation of creative text or even the composition of musical compositions - these are just some of the areas in which artificial intelligence works. Natural language generation is a very promising sub-sector of artificial intelligence and computational linguistics, the main goal of which is to develop computer programs that can create human-perceived text. Among the automatically generated text types, you can even find various weather forecasts, biographies, as well as texts that contain creativity - various stories, jokes, or poetry. The automatic generation of poetry is a seemingly difficult task and a very interesting subject for artificial intelligence research.

The aim of this work "Computer generated poetry" is to show that language models are indeed able to automatically generate texts that can be considered poetic, as well as to use a language model to generate poetry in Latvian. In the course of the work, a review of the theory and a review of the history of computer-generated poetry are made. Various computer-generated text generation algorithms are reviewed and compared. It has been evaluated which model is more suitable for individual use with limited computational resources for poetry generation and its further research has been performed. Based on the literature and the acquired knowledge, a language model has been selected, and the generation of poetry in the Latvian language has been performed.

Keywords: natural language processing, Transformer model, GPT-2 model, computer generated poetry.

AUTOREFERĀTS

Darbā radīti datora ģenerēti dzejas paraugi latviešu valodā. To ģenerēšanai izmantotas latviešu valodai pielāgotas jau esošās idejas no oriģinālā *GPT-2* valodas modeļa, kas ir paredzēts angļu valodai.

Izmantotā literatūra un pielietotie risinājumi galvenokārt ir balstīti uz zinātniskām publikācijām un citu valstu mūsdienu pētnieku novērojumiem šajā jomā. Lielākā daļa no izmantotajiem avotiem ir interneta publikācijas.

Dziļāka izpēte veltīta *Transformer* un *GPT-2* modeļiem, to arhitektūrai un darbībai, jo *GPT-2* modeļa uzbūve lielā mērā balstās uz *Transformer* modeļa arhitektūru.

Apjomīgs darbs veikts datu iegūšanai un apkopošanai, lai veidotu valodas korpusu. Laikietilpīgākais process, protams, bija abu metožu praktiskais pielietojums – gan *GPT-2* modeļa pielāgošanas mēģinājums, gan īstenotā modeļa apmācība.

Darba rezultātā ir izdevies ģenerēt dzeju latviešu valodā, taču tā satur gramatikas kļūdas, tai raksturīga vārdu atkārtošanās un nav izteikti vērojamas atskaņu iezīmes. Apskatot dažādas publikācijas, nākas secināt, ka līdzīgi novērojumi bijuši arī citiem entuziastiem, kas cenšas ģenerēt jebkāda veida tekstu mazāk izplatītās valodās.

Darba teksts ir pārlasīts un tajā nav pareizrakstības kļūdas. Darbā izmantota latviešu valodas terminoloģija – ja oficiālā terminu portālā kādu terminu nav izdevies atrast, tas noteikti ir norādīts darbā iekļautajā apzīmējumu sarakstā.

Visi formulējumi un idejas ir atzīmēti ar attiecīgām literatūras atsaucēm un tiešie citāti ir atzīmēti kā aizguvumi.

SATURS

APZĪMĒJUMU SARAKSTS

NLP – dabiskās valodas apstrāde (*natural language processing*)

Pašuzmanība – angļu val. *self-attention*

RNN – rekurentie neironu tīkli (*recurrent neural network*)

Uzmanības mehānisms – angļu val. *attention mechanism*

LSTM – ilgtermiņa īstermiņa atmiņa (*long short-term memory*)

CNN – konvolūcijas neironu tīkli (*convolutional neural network*)

FFNN – vienvirziena neironu tīkls (*feedforward neural network*)

Vārdlietojuma kartēšanas algoritms – angļu val. *embedding algorithm*

Tekstvienība – angļu val. *token*

Pielāgošana – angļu val. *fine-tuning*

GPU – grafiskais processors *(graphics processing unit)*

TPU – tenzorprocesors *(tensor processing unit)*

IEVADS

Visas valodas izpausmes, vai tas būtu skaļi izteikts vai uzrakstīts teksts, ietver sevī ļoti daudz informācijas – intonācija, kādā cilvēks izsaka vārdus, ietekmē teksta nozīmi, arī dažu īpatnējāku vārdu izvēle vai tas, par kādu tēmu tiek runāts, var cilvēkam palīdzēt vieglāk izprast otra cilvēka sajūtas, noskaņu. Cilvēkiem nav grūti vienam otru saprast (gan vārdiski izteikto, gan uz papīra uzrakstīto), jo jau kopš mazotnes cilvēks mācās uztvert daudz informāciju no apkārtējās pasaules, apstrādā šo iegūto informāciju un spēj izdarīt secinājumus. Lai arī dators spētu izprast dabisko valodu gandrīz kā cilvēks, ir izveidoti dažādi dabiskās valodas apstrādes neironu tīklu modeļi. Šie modeļi principā darbojas līdzīgi kā cilvēka smadzenes – tiek saņemta noteikta informācija, kuru izmantojot, modeļi veido dažādas asociācijas, spēj izvērtēt lietu nozīmību un izdarīt atbilstošus secinājumus, kā arī pieņemt dažādus lēmumus.

Datora ģenerēta dzeja ir aktuāls temats jau kopš aptuveni 1950. gada, kad parādījās pirmie datora ģenerētie literatūras darbi. [1] Mūsdienās, protams, dzejas ģenerēšanas pieejas, izmantojot datoru, atšķiras – vairāk uzmanības tiek pievērsts tieši mākslīgā intelekta izmantošanai, cenšoties tikt galā ar dažādām grūtībām, ar ko pētniekiem nākas saskarties, lai datora ģenerētie darbi būtu pēc iespējas līdzīgāki cilvēka rakstītiem literatūras darbiem.

Lai apzinātu šīs grūtības un saprastu, uz ko jāfokusējas, apmācot modeli uz latviešu valodas literatūras darbiem, darba uzdevumi ir apskatīt dažādas publikācijas par dabiskās valodas apstrādi, teksta ģenerēšanu, kā arī izpētīt dažādas pieejas, kā pielietot valodas modeļus mazāk izplatītām valodām.

Darbā padziļināti pētīts *Transformer* modelis, kā arī *GPT–2* valodas modelis, jo tas tiek izmantots par pamatu mērķa sasniegšanai – dzejas ģenerēšanai latviešu valodā. Valodas modelis tiek apmācīts uz latviešu literatūras darbiem – dzejoļiem, kas iegūti no dažādiem avotiem –, kas veido attiecīgajam uzdevumam pielāgotu valodas korpusu.

1. DABISKĀS VALODAS APSTRĀDE

Dabiskās valodas apstrāde (NLP) ir datorzinātņu, precīzāk, mākslīgā intelekta apakšnozare un tās mērķis ir nodrošināt datoriem iespēju izprast gan rakstisku tekstu, gan izrunātus vārdus tāpat, kā to veic cilvēki.

NLP apvieno datorlingvistiku — likumos balstītu dabiskās valodas modelēšanu — ar statistikas, mašīnmācīšanās un dziļās mašīnmācīšanās modeļiem. Šīs tehnoloģijas dod iespēju datoriem apstrādāt dabisko valodu rakstiska teksta vai balss datu veidā un izprast tās nozīmi, radīto noskaņu, kā arī autora nodomu. [2]

1.1. Dabiskās valodas apstrādes uzdevumi

Dabiskā valoda ir piepildīta ar dažādām valodas īpatnībām, kas ievērojami apgrūtina tādas programmatūras izveidi, kas spēj precīzi noteikt teksta vai balss datu patieso nozīmi. Ir dažādi tēlainās izteiksmes līdzekļi (epiteti, metaforas, salīdzinājumi), emocionāli ekspresīva leksika jeb sarunvalodas vārdi, frazeoloģismi, kā arī vārdi ar atšķirīgu nozīmi, kas tiek izrunāti un rakstīti vienādi, jeb homonīmi. Nereti tiek izmantots arī sarkasms un dažādi gramatikas un lietojuma izņēmumi, teikumu struktūras variācijas — šīs ir tikai dažas no dabiskās valodas īpatnībām, kuru apguvei cilvēkiem ir nepieciešami gadi, taču programmētājiem jāiemāca dabiskās valodas apstrādes programmatūrai tās atpazīt.

Dabiskās valodas apstrādei ir daudz uzdevumi, piemēram, teksta ģenerēšana, teksta klasifikācija un apkopošana, pareizrakstības pārbaude, nosaukto entitāšu atpazīšana, tekstā paustās noskaņas analīze, teikumu dalīšana tekstvienībās (angļu val. – token), atslēgvārdu identificēšana, vārda nozīmes noteikšana, jautājumu un atbilžu ģenerēšana, viltus ziņu noteikšana, surogātpasta identificēšana, runas pārvēršana tekstā un otrādi, dialoga izpratne, mašīntulkošana, aptauju analīze, mērķtiecīgas reklāmas izveide un pat balss asistentu izveide. Dažiem no šiem uzdevumiem ir tieša pielietošana reālajā pasaulē, savukārt citi vairāk darbojas kā apakšuzdevumi, palīdzot risināt lielākas problēmas.

Daži tipiskākie valodas apstrādes uzdevumi:

- Teksta ģenerēšana: Daudziem NLP uzdevumiem ir nepieciešams ģenerēt cilvēkam līdzīgu valodu. Teksta apkopošana un mašīntulkošana principā pārvērš vienu tekstu citā secībā pēc kārtas (*seq2seq* pieeja). Citi uzdevumi, piemēram, attēlu un video apraksti, automātiska laikapstākļu un sporta rezultātu ziņošana, pārvērš netekstuālus datus tekstā. Ir arī daži uzdevumi, kas veido tekstu bez ievades datiem, vai arī ir tikai neliels daudzums datu un tie tiek izmantoti tikai kā tēmas vai virziena norāde – dažādu joku, stāstu un arī dzejas ģenerēšana.
- Mašīntulkošana: Patiesi noderīga mašīntulkošana ietver vairāk nekā vārdu aizstāšanu vienā valodā ar citas valodas vārdiem. Efektīvai tulkošanai ir precīzi jāizprot teksta nozīme un noskaņa ievades valodā un jāpārtulko tekstā ar tādu pašu nozīmi un vēlamo noskaņu arī izvades valodā.
- Nosaukto entitāšu atpazīšana: Nepieciešams atrast un klasificēt entītijas iepriekš noteiktās kategorijās, piemēram, personu vārdi, atrašanās vietas, laika izteiksmes, medicīniskie kodi, organizācijas, daudzumi, naudas vērtības, procenti un citās.
- Dalīšana tekstvienībās: Tas ir veids, kā sadalīt teksta daļu mazākās daļās, ko sauc par tekstvienībām. Tekstvienības var būt vārdi, rakstzīmes vai apakšvārdi. Tā kā tekstvienības ir dabiskās valodas pamatelementi, visizplatītākais neapstrādāta teksta apstrādes veids norisinās tieši tekstvienību līmenī. Piemēram, modeļi, kuru pamatā ir *Transformer* modeļa uzbūve, arī apstrādā neapstrādātu tekstu tekstvienību līmenī.

1.2. Datora ģenerētas literatūras vēsture

Daudzus gadus elektroniskās literatūras kopiena ir uzskatījusi vācu matemātiķa Teo Lutza (vācu val. – Theo Lutz) 1959. gadā publicētos "Stohastiskos tekstus" par pirmo digitālo literāro tekstu. Vācu zinātnieks, filozofs un dzejnieks Makss Benss (vācu val. – Max Bense) ieteica Lutzam izmantot nejaušības ģeneratoru, lai veidotu tekstus. Lutz izveidoja datubāzi ar sešpadsmit priekšmetiem un sešpadsmit nosaukumiem no Franca Kafkas (vācu val. – Franz Kafka) romāna "*Pils*". Lutz programma nejauši ģenerēja skaitļu secību, ieguva katru no priekšmetiem un nosaukumiem, pēcāk savienoja tos, izmantojot loģiskās konstantes (dzimumu, locījumu utt.), lai

izveidotu sintaksi. Viņa projekta rezultāti tika publicēti kā eseja *Augenblick* žurnālā. Publikācija žurnālā ļāva uzskatīt "Stohastiskos tekstus" (vācu val. – *Stochastische Texte*) par pašu pirmo digitālās literatūras tekstu.

Tomēr dažus gadus iepriekš, 1952. gadā, Kristofers Strahejs (angļu val. – Christopher Strachey) izveidoja to, ko drīzāk varētu uzskatīt par pirmo digitālās literatūras tekstu. Strahejs, izmantojot Tjūringa nejaušo skaitļu ģeneratoru, izstrādāja programmu *Mark I,* kas izveidoja mīlestības vēstuļu ģeneratoru "Mīlestības vēstules". Šis bija pirmais digitālās literatūras un digitālās mākslas teksts, kas gandrīz desmit gadus bija pirms agrākajiem digitālās datormākslas piemēriem. Kopš 1960. gada ir veikti daudzi citi eksperimenti datora ģenerētas dzejas jomā, galvenokārt Eiropā, Apvienotajā Karalistē un Amerikas Savienotajās Valstīs. [1]

1.3. Dabiskās valodas apstrādes pieejas

NLP pieejas paver daudzas iespējas cilvēka un datora mijiedarbībai, kas ir tikušas pētītas jau gadu desmitiem. Uz skriptiem balstītas sistēmas, kas spēj "apmānīt" cilvēkus, liekot viņiem domāt, ka viņi runā ar īstu personu, pastāv jau kopš 1970. gadiem [4]. Taču mūsdienu programmas, kas ir aprīkotas ar mašīnmācīšanos un dziļās mašīnmācīšanās algoritmiem, pārsniedz agrākās pieejas un palīdz atrisināt daudzas teksta un runas apstrādes problēmas. Tomēr visas šīs metodes mūsdienās pastāv līdzās, un katrai no tām ir sava jēga noteiktos lietošanas gadījumos.

1.3.1. Likumos balstīta dabiskās valodas apstrāde

Šī pieeja ir lieliski piemērota datu pirmapstrādei. Likumi tiek uzskatīti par novecojušu pieeju teksta apstrādei. Tie ir rakstīti manuāli un nodrošina ikdienas uzdevumu pamata automatizāciju. Piemēram, var rakstīt likumus, kas ļaus sistēmai tekstā identificēt e-pasta adresi, jo tai ir pazīstams formāts, taču, tiklīdz tiek ieviests kāds jauninājums, sistēmas iespējas beidzas līdz ar likumu autora zināšanām. Tomēr likumos balstīta dabiskās valodas apstrāde joprojām tiek izmantota šodien, jo atsevišķos gadījumos šī pieeja ir pietiekami efektīva. Viens no gadījumiem ietver uzdevumus,

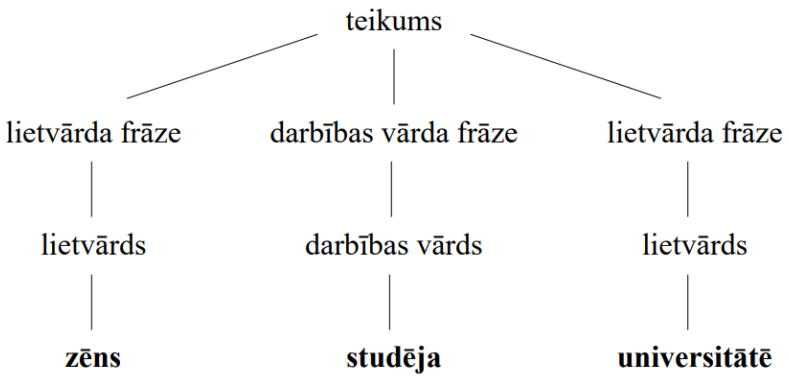
kuriem jau eksistē likumu bāze. Piemēram, gramatikā ir savs likumu kopums, tas pats attiecas arī uz pareizrakstību. Sistēma, kas ir aprīkota ar vārdnīcu, savu darbu veiks labi, bet tā nevarēs ieteikt labāku vārdu vai frāzes izvēli. Arī gadījumos, kad likumu kopums ir neliels, likumos balstīta dabiskās valodas apstrāde ir atbilstošs risinājums, taču tiklīdz parādās simtiem likumu, tie sāk mijiedarboties neparedzētā veidā, un šādas sistēmas apkope, visticamāk, nav tā vērta.

1.3.2. Uz mašīnmācīšanos balstīta dabiskās valodas apstrāde

Mašīnmācīšanās jeb statistiskā NLP pieeja ietver mākslīgā intelekta algoritmu izmantošanu, lai atrisinātu problēmas bez tiešas programmēšanas. Tā vietā, lai strādātu ar cilvēku rakstītiem modeļiem, mašīnmācīšanās modeļi atrod šos šablonus (angļu val. – patterns) tikai analizējot tekstus. Ir divas galvenās darbības, lai sagatavotu datus tā, lai mašīna tos saprastu. Jebkurš mašīnmācīšanās projekts sākas ar datu sagatavošanu – nepieciešams veikt teksta anotāciju un formatēšanu. NLP uzdevumos šo procesu sauc par korpusa izveidi. Korpusi ir tekstu krājumi, ko izmanto mašīnu apmācībā. Nepietiek, ja vienkārši sistēmā tiek ievadīta, piemēram, visa e-pasta datu kopa – sistēma nesapratīs, ko no tās vēlas. Tāpēc tekstiem ir jābūt anotētiem — tie jāpastiprina, piešķirot tiem lielāku nozīmi. Dalīšana tekstvienībās, runas daļas marķēšana un teikuma sintaktiskā analīze, kas apraksta teikuma sintaktisko struktūru, ir daži no anotāciju veidiem.

Teikuma sintaktiskā analīze ir svarīgs mašīnmācīšanās process dabiskajā valodas apstrādē, jo modeļiem nepieciešams labi izprast teikuma nozīmi no dabiskās valodas teksta vai runas. Teikumu sintaktiskā analīze cilvēku smadzenēs notiek pastāvīgi, pašiem cilvēkiem to pat nemanot. Lasot jebkuru tekstu, cilvēka smadzenēs arī notiek šis process, kurā ievades plūsma tiek pārveidota par kādu strukturētu attēlojumu. Ievades plūsma var sastāvēt gan no vārdiem, gan rakstzīmēm. Šī procesa rezultātā tiek iegūts sintaktiskās analīzes koks.

Cilvēka smadzenes ļoti labi prot analizēt tekstu. Kad mēs dzirdam tādu teikuma daļu kā piemēram “zēns studēja universitātē”, mūsu smadzenes izveido sintaktiskās analīzes koku, kas ir līdzīgs 1.1. attēlā redzamajam.



1.1. att. **Vienkārša teikuma sintaktiskā analīze**

Pēc tam anotētie dati tiek sakārtoti standarta formātos, tādējādi kļūstot strukturēti.

Otra galvenā darbība ir raksturiezīmju inženierija. Papildus korpusam mašīnas izmanto dažādas funkcijas teksta uztveršanai. Funkcijas ir dažādas īpašības, piemēram, vārdu skaits, pieturzīmju skaits vai vārdu biežums, kas var pateikt sistēmai, kas tekstā ir svarīgs. Datu zinātniekiem ir uzdevums izlemt, kuras teksta iezīmes palīdzēs modelim atrisināt problēmu. Piemēram, biežuma funkcija vārdiem "tagad", "nekavējoties", "bez maksas" un "zvans" var norādīt, ka ziņojums ir surogātpasts, bet pieturzīmju skaitīšanas funkcija var norādīt uz pārmērīgu izsaukuma zīmju lietošanu.

Mašīnmācībai ir nepieciešams, lai datu zinātnieki izstrādātu jau iepriekš minētās funkcijas. Problēma ir tā, ka valodās ir simtiem tūkstošu vārdu. Vārdu savienojumu skaitu, kas varētu veidot noteiktas nozīmes vai, piemēram, visu iespējamo teikumu skaitu nav iespējams izskaitīt. "Pat ar milzīgu piemēru kopu mēs, ļoti iespējams, novērojam notikumus, kas nekad nav notikuši piemēru kopā un kas ļoti atšķiras no visiem piemēriem, kas tajā notika," atzīmē Bar – Ilana universitātes (*Bar Ilan University*) profesors Joavs Goldbergs (*Yoav Goldberg*). [3] Tātad, apstrādājot valodas datus ar mašīnmācīšanos, rodas datu nepietiekamības problēma. Valodas datu nepietiekamība ir saistīta ar citu problēmu, ko datu zinātnieki sauc par dimensiju lāstu. [5] Jo vairāk pazīmju (dimensiju) datu kopā, jo lielāks datu apjoms ir jāvispārina. Tas nozīmē lielāku slodzi mašīnmācīšanās algoritmam. Tā kā mašīnmācīšanās modeļi nevar efektīvi apstrādāt datus ar tik daudziem atribūtiem, ir nepieciešams pielietot citu metodi.

1.3.3. Uz dziļo mašīnmācīšanos jeb uz neironu tīkliem balstīta dabiskās valodas apstrāde

Dziļā mašīnmācīšanās ir nozare, kas koncentrējas uz dziļo neironu tīklu apmācību. Dziļie neironu tīkli tiek tā saukti, jo tiem ir vairāki slēpti slāņi, un to veiktspēja pieaug līdz ar to skaitu. Viena no labākajām dziļās mašīnmācīšanās priekšrocībām ir tā, ka tā ļauj pētniekiem mazāk veikt manuālu darbu, jo neironu tīklu modeļi patstāvīgi var apgūt funkcijas no apmācības datiem, tiem nav nepieciešama uzdevumam specifiska funkciju izstrāde. Dziļās mašīnmācīšanās pieejas ātri pārspēja statistiskās mācīšanās metodes ar ievērojami labākiem rezultātiem. Šobrīd uz neironu tīkliem balstītā pieeja ir sasniegusi jaunus kvalitātes līmeņus un kļuvusi par dominējošo pieeju NLP uzdevumiem. Jauni dziļās mašīnmācīšanās modeļi tiek ieviesti arvien biežāk un viens konkrēts neironu tīklu modelis ir izrādījies īpaši efektīvs tipiskos dabiskās valodas apstrādes uzdevumos – tas ir *Transformer* modelis.

2. TRANSFORMER MODELIS

Transformer modelis ir dziļas mašīnmācīšanās modelis – tas izmanto pašuzmanības (angļu val. – self-attention) mehānismu, lai atšķirīgi spētu novērtēt katras ievaddatu daļas nozīmi. *Transformer* modeli galvenokārt izmanto dabiskās valodas apstrādes un arī datorredzes jomā.

Transformer modelis, tāpat kā rekurentie neironu tīkli (RNN), ir paredzēts, lai apstrādātu secīgus ievades datus, bet, atšķirībā no RNN, *Transformer* modeļi ne vienmēr apstrādā datus noteiktā secībā. *Transformer* modelī esošais uzmanības mehānisms (angļu val. – attention mechanism) sniedz kontekstu jebkurai pozīcijai ievades secībā. Piemēram, ja ievades dati ir dabiskās valodas teikums, *Transformer* modelim nav nepieciešams obligāti apstrādāt teikuma sākumu pirms teikuma beigām. Modelis identificē kontekstu, tādā veidā spējot piešķirt nozīmi katram vārdam teikumā. Šī funkcija nodrošina labāku paralēlo skaitļošanu (modeļa slāņu sadalīšanu vairākos grafiskajos procesoros) nekā RNN – pateicoties tam, samazinās modeļa apmācības laiks.

Transformer modeļus 2017. gadā ieviesa *Google Brain* komanda, un tie aizvien biežāk tiek izvēlēti NLP problēmu risināšanai, aizstājot RNN modeļus. Paralēlā skaitļošana sniedz iespēju apmācīt modeļus ar lielākām datu kopām, nekā tas bija iespējams agrāk. Tā rezultātā ir izstrādātas iepriekš apmācītas sistēmas, piemēram, *BERT* (Bidirectional Encoder Representations from Transformers) un *GPT* (Generative Pre-trained Transformer), kuras tika apmācītas ar lielām valodu datu kopām, piemēram, *Wikipedia Corpus* un *Common Crawl,* un pēcāk šie modeļi var tikt pielāgoti citiem, specifiskākiem uzdevumiem. [6]

2.1. Modeļa attīstība

Pirms *Transformer* modeļiem lielākā daļa jaunāko NLP sistēmu tika balstītas uz RNN modeļiem, kas papildināti ar uzmanības mehānismiem, toties *Transformer* modeļi ir veidoti, neizmantojot RNN struktūru, bet balstoties uz uzmanības mehānismiem, uzsverot faktu, ka pat tikai ar uzmanības mehānismiem ir iespējams panākt to pašu veiktspēju kā RNN modeļiem ar

papildu uzmanības mehānismiem.

RNN gadījumā iepriekšējo darbību izvade tiek ievadīta pašreizējā stāvokļa ievadē. Lai varētu paredzēt jebkura vārda nākamo burtu vai teikuma nākamo vārdu, modelim ir jāatceras iepriekšējie burti vai vārdi un jāsaglabā tie atmiņā. Teorētiski informācija no vienas tekstvienības var patvaļīgi izplatīties tālāk sekvencē, ja stāvoklis katrā punktā turpina kodēt kontekstuālo informāciju par tekstvienību. Realitātē šis mehānisms ir kļūdains: izzūdošā gradienta problēma atstāj modeļa stāvokli gara teikuma beigās bez iespējas iegūt precīzu informāciju par iepriekšējām tekstvienībām. Tekstvienību aprēķinu atkarība no iepriekšējo tekstvienību aprēķinu rezultātiem apgrūtina arī iespēju veikt paralēlo skaitļošanu ar dziļās mašīnmācīšanās aparatūru – tas viss var padarīt RNN modeļu apmācību ne pārāk efektīvu. [7]

Iepriekš minētās problēmas tika risinātas ar uzmanības mehānismiem. Uzmanības veidošanas mehānismi ļauj modelēt stāvokli jebkurā iepriekšējā punktā visā sekvencē. Uzmanības slānim ir iespēja piekļūt visiem iepriekšējiem stāvokļiem un nosvērt tos atbilstoši apgūtajam atbilstības mērījumam, tādējādi sniedzot būtisku informāciju arī par tālākām tekstvienībām.

Labs piemērs tam, cik vērtīgi ir uzmanības mehānismi, ir teksta tulkošana, kur svarīgs ir viss kopējais teksta saturs, lai varētu piešķirt vārda nozīmi teikumā. Tulkošanas sistēmā no, piemēram, angļu valodas uz latviešu valodu pirmais vārds latviešu valodā, visticamāk, būtu atkarīgs no angļu valodas ievades dažiem pirmajiem vārdiem. Klasiskajā ilgtermiņa īstermiņa atmiņas (LSTM) modelī, lai izveidotu latviešu valodas izvades pirmo vārdu, modelim tiktu dots tikai pēdējā angļu valodas vārda stāvokļa vektors. Teorētiski šis vektors var iekodēt informāciju par visu angļu valodas teikumu, tādējādi nodrošinot modelim visas nepieciešamās zināšanas. Realitātē LSTM šo informāciju bieži vien slikti saglabā. Arī šīs problēmas risināšanai var pievienot uzmanības mehānismu – dekodētājam tiek nodrošināta piekļuve katra angļu valodas ievades vārda stāvokļa vektoriem, ne tikai pēdējam, un tas apgūst uzmanības svarus, kas, savukārt, nosaka, cik daudz uzmanības jāpievērš katram angļu valodas ievades stāvokļa vektoram.

Transformer modeļu arhitektūras veidošanas process atklāja, ka uzmanības mehānismi paši par sevi ir spēcīgi un ka nemaz nav nepieciešama secīga atkārtota datu apstrāde, lai ar uzmanības mehānismiem sasniegtu RNN modeļu veiktspēju. *Transformer* modeļi izmanto uzmanības mehānismu bez RNN, vienlaikus apstrādājot visas tekstvienības un secīgos slāņos aprēķinot uzmanības svaru starp tām. Tā kā uzmanības mehānisms izmanto informāciju tikai par citām tekstvienībām no zemākiem slāņiem, to var aprēķināt visām tekstvienībām paralēli un nodrošināt

paralēlo skaitļošanu, tādējādi uzlabojot modeļa apmācības ātrumu.

Dažas no iepriekš minētajām problēmām joprojām nav atrisinātas ar RNN modeļiem, kas papildināti ar uzmanības mehānismu. Piemēram, nav iespējams nodrošināt paralēlo skaitļošanu ievades apstrādei – apjomīgam teksta korpusam tas palielina apstrādei patērēto laiku.

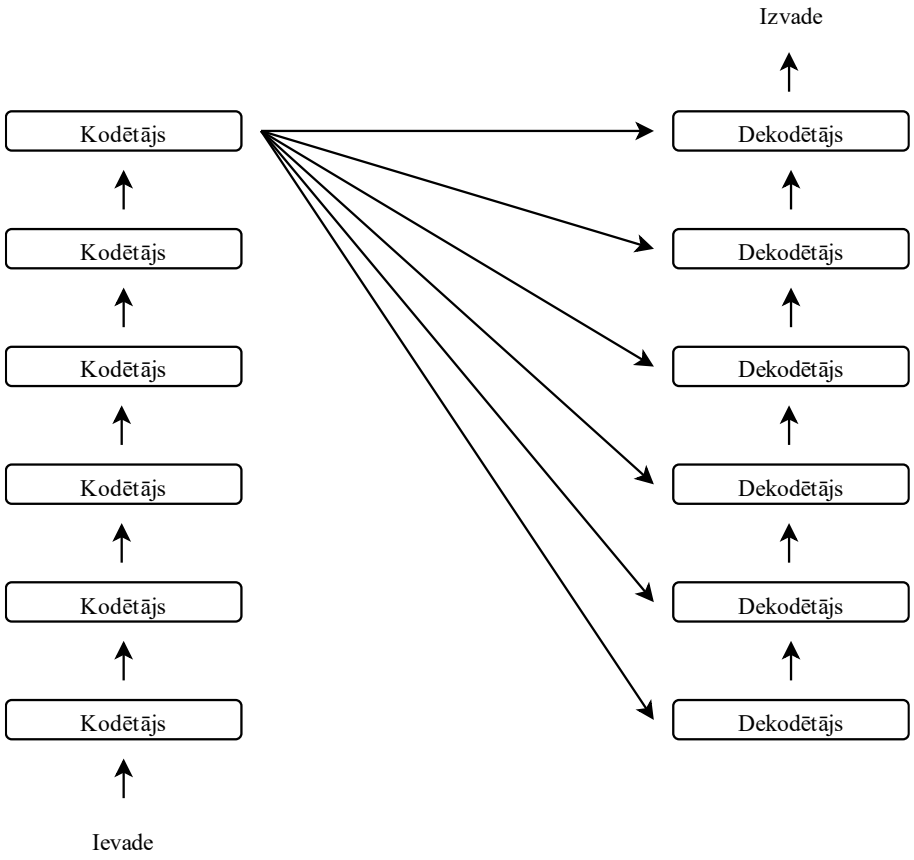
Konvolūcijas neironu tīkli (CNN) palīdz atrisināt šīs problēmas – ar tiem var veikt paralēlo skaitļošanu, jo katru ievades vienību var apstrādāt vienlaikus (apstrāde ne vienmēr ir atkarīga no iepriekšējām vienībām).

Problēma ir tā, ka CNN ne vienmēr tiek galā ar atkarību problēmu dabiskās valodas apstrādes uzdevumos, kā arī gadījumos, kad nepieciešams apskatīt garākus teikumus, CNN modeļu gadījumā ir nepieciešams dziļš tīkls. Šīs problēmas palīdz risināt *Transformer* modeļi, kas veidoti kā CNN kombinācijā ar uzmanības mehānismiem.

2.2. Modeļa arhitektūra

Lai atrisinātu iepriekš minētās problēmas, *Transformer* modelis ir veidots, izmantojot CNN kopā ar uzmanības mehānismiem. Uzmanības mehānismi uzlabo to, cik ātri modelis var pāriet no vienas sekvences uz citu. Lai palielinātu šo ātrumu, modelis izmanto uzmanības, precīzāk, pašuzmanības mehānismu.

Transformer modelim ir līdzīga arhitektūra kā iepriekš minētajiem modeļiem. Modelī ir divas komponentes – kodētājs un dekodētājs. Ilustratīva modeļa komponenšu uzbūve ir apskatāma 2.1. attēlā. [8]

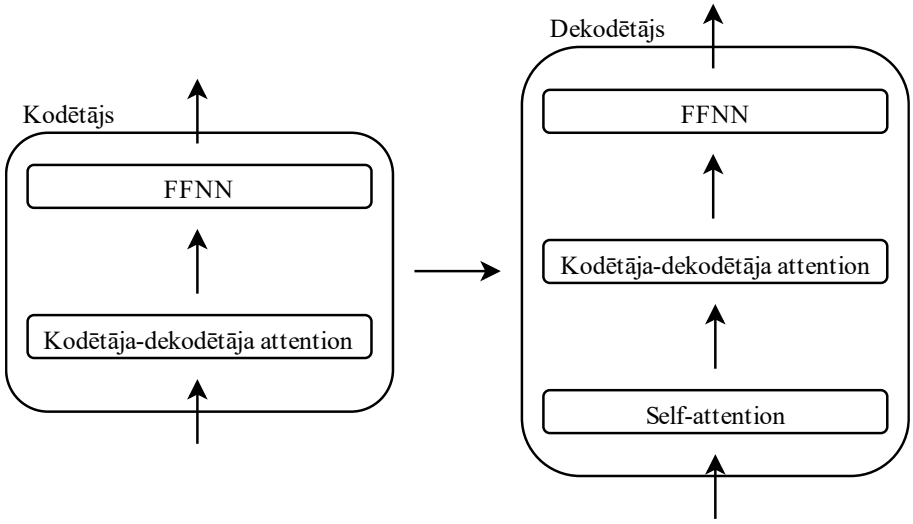


2.1. att. **Transformer modeļa komponentes**

Kodētāja bloki ir savstarpēji ļoti līdzīgi – tiem visiem ir vienāda arhitektūra. Katrs kodētāja bloks satur divus slāņus: pašuzmanības slāni un vienvirziena neironu tīkla (FFNN) slāni (sk. 2.2. att.). Kodētāja ievades dati iesākumā virzās caur pašuzmanības slāni, tādā veidā ļaujot kodētājam katra vārda apstrādes brīdī apskatīt arī citus ievades vārdus teikumā. Tas palīdz labāk izprast vārda nozīmi attiecīgajā kontekstā. Izvade no pašuzmanības slāņa tiek virzīta tālāk uz FFNN slāni. [8]

Arī dekodētāja bloki ir ļoti līdzīgi viens otram un katrs dekodētāja bloks, tāpat kā katrs kodētāja bloks, satur abus iepriekšminētos slāņus (pašuzmanības un FFNN), bet starp tiem papildus atrodas uzmanības slānis (sk. 2.2. att.). Pašuzmanības slānī dekodētāja ievadē dažas pozīcijas tiek maskētas, tātad šajā slānī tās tiek ignorētas. Paredzot teikuma nākamo vārdu, dekodētājam nevajadzētu zināt, kurš vārds seko aiz paredzētā vārda. Dekodētājam ir jāzina tikai vārdi līdz pašreizējai pozīcijām. Tas tiek darīts tāpēc, lai modeļa treniņa laikā dekodētājam nebūtu iespēja "krāpties", atvieglojot sev darbu – arī realitātē izmantojot modeli, lai prognozētu nākamo vārdu, dekodētājam nav iespējas redzēs nākamās pozīcijas. Otrs dekodētāja slānis – uzmanības slānis – ir

paredzēts, lai nodotu informāciju no kodētāja uz dekodētāju. Vaicājuma vektors tiek iegūts no iepriekšējā kodētāja, bet atslēgas un vērtības vektors no augstākā kodētāja izvades. Tas nodrošina iespēju, ka dekodētājs var ņemt vērā visas pozīcijas kodētāja ievades sekvencē. [9]

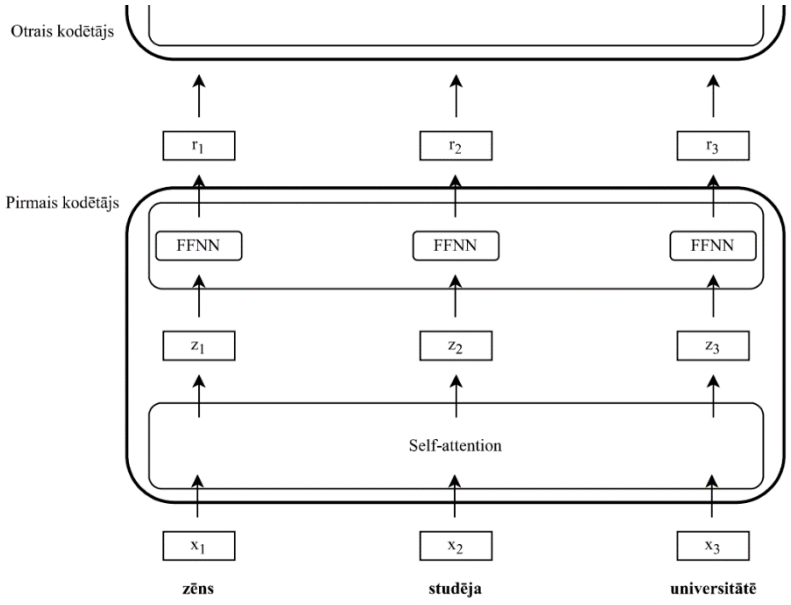


2.2. att. **Kodētāja un dekodētāja bloku uzbūve Transformer modelī**

Tā kā dabiskās valodas apstrādē ieejas dati ir vārdi, sākumā katrs ievades vārds tiek pārvērsts par vektoru, izmantojot vārdlietojuma kartēšanas algoritmu (angļu val. – *embedding algorithm*). Vārdlietojuma kartēšanas algoritms atsevišķus vārdus attēlo kā reālas vērtības vektorus (tie var būt ar desmitiem vai pat simtiem dimensiju) iepriekš noteiktā vektoru telpā – katrs vārds tiek kartēts uz vienu vektoru. Izkliedētā reprezentācija tiek apgūta, pamatojoties uz vārdu lietojumu. Tas nozīmē, ka vārdi, kas tiek pielietoti līdzīgā veidā, iegūst līdzīgus attēlojumus, savukārt tas dod iespēju modelim uztvert šo vārdu nozīmi. [10]

Katrs no kodētājiem kā ievades datus saņem vektoru sarakstu, taču tikai pirmajā kodētājā notiek vārdlietojuma kartēšana – pārējos kodētājos tiek saņemta iepriekšējā kodētāja izvades dati. Pēc vārdlietojuma kartēšanas ievades sekvencē katrs no vārdiem plūst cauri diviem kodētāja slāņiem (sk. 2.3. att.). [11] Katrā pozīcijā vārds kodētājā plūst pa savu ceļu. Pašuzmanības slānī šie dažādie ceļi ir savstarpēji atkarīgi, bet FFNN slānī nav šo atkarību, tādēļ dažādie ceļi var tikt izpildīti paralēli brīdī, kad tie plūst caur FFNN slāni.

Kodētāja pašuzmanības slānī dotā ievades simbolu vektoru kopa $x = (x_1, \dots, x_n)$ tiek pārveidota par nepārtrauktu secīgu attēlojumu $z = (z_1, \dots, z_n)$. Ņemot vērā no pašuzmanības slāņa saņemtos ievades datus z, FFNN slānis ģenerē secīgu simbolu attēlojumu pa vienam elementam $(r_1, \dots, r_n)$, kas tālāk tiek sūtīts nākamam kodētājam (sk. 2.3. att.). [11]



2.3. att. **Datu plūsma pa kodētāja komponentēm**

2.3. Transformer modeļa pašuzmanības mehānisms

Ir zināms, ka vārdu kartējumi ir vektori, kas atspoguļo vārda semantisko nozīmi. Vārdiem ar līdzīgu nozīmi var būt līdzīgi kartējumi. Tomēr vārdu atsevišķās nozīmes neatspoguļo to nozīmi teikumā. Piemēram, ja teikumā ir vārdi "Latvijas Universitāte", vārda "Latvijas" un vārda "Universitāte" kartējumi atsevišķi nozīmē dažādas lietas, bet attiecīgajā teikumā tiem ir spēcīga korelācija, kas cilvēkam ir viegli uztverams, bet datoram tas var sagādāt grūtības. Vārdu kartēšanai bez pašuzmanības mehānisma nepiemīt spēja noteikt attiecīgā konteksta sajūtu, tāpēc, ņemot vērā iepriekš minēto frāzi, valodas modelim nav liela iespēja paredzēt vārdu "Universitāte" pēc vārda "Latvijas". Lai risinātu šo problēmu, kā daļa no sākotnējās *Transformer* modeļa arhitektūras tika piedāvāts pašuzmanības slānis. [6]

Pašuzmanības slānis *Transformer* modelī ir viena no svarīgākajām komponentēm – tajā notiek procesi, kas ir atbildīgi par vārdu izprašanu tieši konteksta nozīmē. Kad tiek veikta modeļa apmācība, tiek sagatavotas dažādas matricas, kas pēcāk nepieciešamas modeļa darbināšanai – matrica W^{wm} ir vektoros iestrādāto vārdu matrica. Tā satur vektoriālu informāciju par katru vārdu, kas modelim ir zināms no valodas korpusa, kas izmantots modeļa apmācībai. Pieņemot, ka modelim zināmo tekstvienību skaits no valodas korpusa ir N un vārdu vektoru dimensija ir D, matricas W^{wm} izmērs ir $N \times D$. Jo lielāka ir vārda vektora dimensija, jo plašāka informācija par attiecīgo vārdu ir iestrādāta vektorā. Modeļa apmācības laikā tiek sagatavota arī matrica W^{pm}, kas ir pozicionālā matrica – tā tiek izmantota, lai brīdī, kad tiek apstrādāti ieejas dati, tiktu ņemta vērā arī vārdu secība. Pieņemot, ka y ir izmantoto ievades vārdu skaits katra konkrētā vārda apstrādei, matricas W^{pm} izmērs ir $y \times D$. Modeļa darbināšanai un skaitļošanai nepieciešamas vēl trīs matricas, arī tās tiek sagatavotas apmācības brīdī – atslēgu svaru matrica W^k, vaicājumu svaru matrica W^q un vērtību svaru matrica W^v. [17]

Pirmais kodētāja bloks kā ievades datus saņem secīgu vārdu rindu $w_1, w_2, \dots, w_n$, kur n ir ievades vārdu skaits. Katram ieejas vārdam w_i, kur $i = 1, \dots, n$, no matricas W^{wm} tiek atrasts atbilstošs vārdu vektors $\vec{x}_i$. Katram ieejas vārdam atkarībā no atrašanās vietas tekstā tiek atrasts arī vektors $\vec{t}_i$ no matricas W^{pm}. Tālāk pirmā kodētāja pašuzmanības slānis par ievades datiem saņem vektorus $\vec{z}_i = (z_{i1}, z_{i2}, \dots, z_{iD})$, kas veidojas no iepriekš minēto vektoru summas:

$$\vec{z}_i = \vec{x}_i + \vec{t}_i = (x_{i1} + t_{i1}, x_{i2} + t_{i2}, \dots, x_{iD} + t_{iD}). \quad (2.1)$$

Pašuzmanības apakšslānī katram saņemtajam vektoram $\vec{z}_i$ tiek atrasts atslēgas vektors $\vec{k}_i = k_{i1}, k_{i2}, \dots, z_{im}$, vaicājuma vektors $\vec{q}_i = (q_{i1}, q_{i2}, \dots, q_{im})$, kā arī vērtības vektors $\vec{v}_i = (v_{i1}, v_{i2}, \dots, v_{im})$:

$$\vec{k}_i = \vec{z}_i W^k; \quad (2.2)$$

$$\vec{q}_i = \vec{z}_i W^q; \quad (2.3)$$

$$\vec{v}_i = \vec{z}_i W^v. \quad (2.4)$$

Ņemot vērā to, ka modelī vienlaicīgi tiek ievadīti vairāki secīgi ieejas dati, tad iespējams pārvērst vektorus matricu formā:

$$Z = \begin{matrix} \vec{z}_i \\ \vdots \\ \vec{z}_n \end{matrix} = \begin{pmatrix} z_{11} & \cdots & z_{1D} \\ \vdots & \ddots & \vdots \\ z_{n1} & \cdots & z_{nD} \end{pmatrix}; \tag{2.5}$$

$$K = \begin{matrix} \vec{k}_i \\ \vdots \\ \vec{k}_n \end{matrix} = \begin{pmatrix} k_{11} & \cdots & k_{1m} \\ \vdots & \ddots & \vdots \\ k_{n1} & \cdots & k_{nm} \end{pmatrix}; \tag{2.6}$$

$$Q = \begin{matrix} \vec{q}_i \\ \vdots \\ \vec{q}_n \end{matrix} = \begin{pmatrix} q_{11} & \cdots & q_{1m} \\ \vdots & \ddots & \vdots \\ q_{n1} & \cdots & q_{nm} \end{pmatrix}; \tag{2.7}$$

$$V = \begin{matrix} \vec{v}_i \\ \vdots \\ \vec{v}_n \end{matrix} = \begin{pmatrix} v_{11} & \cdots & v_{1m} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nm} \end{pmatrix}. \tag{2.8}$$

Tagad arī pašuzmanības slāņa izejošos vektorus iespējams definēt matricu formā:

$$SA_{out} = \begin{matrix} \overrightarrow{sa}_{1\ out} \\ \vdots \\ \overrightarrow{sa}_{n\ out} \end{matrix} = softmax\left(\frac{QK^T}{\sqrt{m}}\right)V. \tag{2.9}$$

Veicot ievades vārdu apstrādi, tiek aplūkoti arī visi pārējie ievades vārdi. Lai varētu noteikt, cik spēcīgi pārējie vārdi ir saistīti ar apstrādājamo vārdu, tie tiek novērtēti tieši attiecībā pret attiecīgo vārdu. Šo novērtējumu apzīmē ar p_k, kur k ir novērtējamā vārda pozīcija:

$$p_k = \sum_{i=1}^{n} \frac{k_{ki} q_{ki}}{\sqrt{m}} = \frac{k_{k1}q_{k1} + k_{k2}q_{k2} + \cdots + k_{kn}q_{kn}}{\sqrt{m}}. \tag{2.10}$$

Gan katram apstrādājamam vārdam atšķiras novērtējuma p_k vērtība, gan tam apkārt esošo vārdu novērtējuma p_k vērtības atšķiras, tāpēc vārdu novērtējumu katram apstrādājamam vārdam ir iespējams definēt kā vektoru $\vec{p}_i = (p_{i1}, p_{i2}, \dots, p_{in})$. Tālāk nepieciešams normalizēt iegūtās vārdu novērtējumu vērtības, lai izveidotu varbūtisku sadalījumu, kur attiecīgi izpildās:

$$\sum_{j=1}^{n} p_{ij} = 1. \tag{2.11}$$

Lai varētu normalizēt vārdu novērtējumu vērtības, tiek izmantota *softmax* funkcija, kas pārvērš šīs vērtības no svērtās summas vērtībām par varbūtībām, kuru summa ir viens:

$$\vec{p}_{i\,norm} = softmax(\vec{p}_i) = \frac{e^{p_i}}{\sum_{j=1}^{n} e^{p_j}}. \quad (2.12)$$

Pēc tam, kad iegūti normalizētās vārdu novērtējumu vērtības $\vec{p}_{i\,norm}$, tās tiek reizinātas ar vārdu vērtību vektoriem un summētas, tādā veidā tiek iegūti no pašuzmanības slāņa izejošie vektori:

$$\overrightarrow{sa}_{i\,out} = \sum_{j=1}^{n} \vec{p}_{i\,norm\,j}\,\vec{v}_i = \vec{p}_{i\,norm1}\vec{v}_1 + \vec{p}_{i\,norm2}\vec{v}_2 + \cdots + \vec{p}_{i\,normn}\vec{v}_n, \quad (2.13)$$

kur n ir ievades vārdu skaits, bet i ir apstrādājamais vārds, $i = 1, \dots, n$. No pašuzmanības slāņa izejošie vektori $\overrightarrow{sa}_{i\,out}$ pēcāk kļūst par FFNN apakšslāņa ievadi, savukārt no šī apakšslāņa izejošie vektori kļūst par ievades datiem nākamajā modeļa kodētāja blokā. Tur norisinās identisks process, tikai ar jau citiem modeļa apmācības procesā iegūtiem parametriem. Process turpinās līdz pēdējam blokam (sk. 2.3. att.), un no tā FFNN apakšslāņa izejošos vektorus $\overrightarrow{ffnn}_{i\,out}$ nepieciešams reizināt ar matricu W^{wm}:

$$\overrightarrow{pw}_i = \overrightarrow{ffnn}_{i\,out} W^{wm}. \quad (2.14)$$

Iegūtie gala vektori $\overrightarrow{pw}_i = pw_{i1}, pw_{i2}, \dots, pw_{iN}$, kur $i = 1, \dots, n$, n ir apstrādājamo vārdu skaits, N ir modelim zināmo vārdu skaits no valodas korpusa, norāda varbūtību sadalījumu tiem vārdiem, kas ir iekļauti modelim zināmajā valodas korpusā. Tādējādi $\max_j pw_{ij}$ nosaka, ka vārdam no matricas W^{wm}, kas atrodas $j - taj$ā pozīcijā, kur $j = 1, \dots, N$, ir lielākā varbūtība kļūt par modeļa izvadi apstrādājamam vārdam i. [11][18]

Transformer modeļa arhitektūrā pašuzmanība tiek attiecināta uz visu apkārtējo kontekstu, piemēram, visi pārējie vārdi teikumā, taču dekodētāja bloku gadījumā tā vietā tiek izmantots maskētās pašuzmanības mehānisms, kas nozīmē, ka dekodētājā pašuzmanības slānim ir atļauts ņemt vērā tikai iepriekšējās tekstvienības izvades sekvencē, maskējot nākamās pozīcijas pirms *softmax* funkcijas izmantošanas pašuzmanības aprēķinos.

2.4. Transformer modeļu salīdzinājums

Transformer modeļi aizvien biežāk tiek izvēlēti NLP problēmu risināšanai – ir izstrādāti iepriekš apmācīti modeļi, piemēram, plaši pazīstamie *BERT* un *GPT*, kuru darbības pamatā ir to apmācība lielām valodu datu kopām, lai pēcāk šie modeļi var tikt pielāgoti citiem, specifiskākiem uzdevumiem.

BERT modeļa priekšrocība ir tā, ka tas izmanto divvirzienu mācīšanos – tas spēj saprast vārdu kontekstu teikumā, tekstu lasot vienlaicīgi abos virzienos – gan no kreisās uz labo pusi, gan otrādi. Atšķirībā no *BERT*, *GPT* modeļi izmanto vienvirziena mācīšanos, taču to priekšrocība ir apjomīgais vārdu apjoms, uz kuriem tas ir iepriekš apmācīts. Tas ļauj lietotājiem pielāgot modeli specifiskākiem NLP uzdevumiem, izmantojot salīdzinoši maz piemēru. Lai ģenerētu tekstu, *GPT* izmanto tikai *Transformer* modeļa arhitektūras dekodētāja daļu (tas balstās uz iepriekšējām vērtībām, lai prognozētu pašreizējās vērtības). Savukārt *BERT* modelis balstās uz visapkārt esošajām vērtībām, nevis tikai uz vērtībām pirms vai pēc aplūkotās vērtības. [12]

GPT ir autoregresīvs modelis, tātad tā uzdevums ir prognozēt nākamo tekstvienību, izlasot visas iepriekšējās. Precīzi pielāgojot modeli, ir iespējams to pielietot un iegūt apmierinošus rezultātus dažādos NLP uzdevumos, labākais pielietojums tam ir tieši teksta ģenerēšana. [13]

BERT modelis ir iepriekš apmācīts kādā noteiktā veidā sabojājot ievades tekstvienības un mēģinot iegūt sākotnējos ievades teikumus. Arī *BERT* modeli var pielāgot citiem NLP uzdevumiem, piemēram, teksta ģenerēšanai un sasniegt apmierinošus rezultātus, taču tā dabiskākais pielietojums ir teikumu vai tekstvienību klasifikācija. [13] Arī rakstā, kur pirmo reizi autori iepazīstina ar *BERT* modeli, ir uzsvērti divi galvenie pielietojumi modelim – atbilžu sniegšana uz jautājumiem un valodas slēdzienu (angļu val. – language inference) noteikšanai, kas būtībā ir uzdevums, lai noteiktu, vai hipotēze ir patiesa, nepatiesa vai tā tiek noteikta par neitrālu. [14]

Izprotot abiem modeļiem piemērotākos NLP uzdevumus, kā arī citu pētnieku pieredzes dzejas ģenerēšanā, tiek izdarīts secinājums, ka dzejas ģenerēšanai piemērotāki ir *GPT* modeļi. *GPT-2* ir atvērtā pirmkoda rīks, kas lietotājiem ir viegli pieejams, lai varētu to pielāgot savām vajadzībām un atrisinātu dažādus NLP uzdevumus. Protams, jaunākais *GPT* modelis – *GPT-3* – ir

labāk piemērots dažādiem NLP uzdevumiem un sniedz labākus rezultātus – tas var apstrādāt citas valodas (kas nav angļu valoda) labāk nekā *GPT-2*. Citi pētnieki ir jau izmēģinājuši izpildīt NLP uzdevumus dažādās valodās, piemēram, vācu, krievu, kā arī japāņu valodā, un tas darbojās apmierinoši. *GPT-3* ir vēl labāk piemērots dzejoļu ģenerēšanai – arī tādu, kas jāveido noteiktā stilistikā. [15] Diemžēl pašlaik *GPT-3* modelim nav atvērtā pirmkoda. Tam ir ierobežota piekļuve un tas nav komerciāli pieejams. *Beta* versijai piekļuve tiek piešķirta ierobežotam lietotāju lokam pēc pieprasījuma.

3. GPT-2 MODELIS

GPT-2 ir *OpenAI* izstrādāts dabiskās valodas apstrādes modelis, kas ir iepriekš apmācīts un kura pamatā ir nepārraudzītas mašīnmācīšanās metodes. Modelis spēj pabeigt un ģenerēt tekstu, lasīt un saprast tekstu, to apkopot, pārrakstīt un pat atbildēt uz jautājumiem par tajā esošo informāciju.

GPT-2 ir *GPT*, sākotnējā *OpenAI* dabiskās valodas apstrādes ietvara, papildinājums. Gan parametru skaits, gan apmācības datu kopas lielums šajā modelī palielināts desmitkārtīgi.

Iepriekš apmācītais modelis ir ar 1,5 miljardiem parametru un tas apmācīts 8 miljonu tīmekļa lapu datu kopā. [16]

Valodas modelēšana parasti ir neuzraudzīta sadalījuma novērtējums no piemēru kopas $(x_1, x_2, \dots, x_n)$, kur katra kopa satur dažāda garuma secīgu simbolu virknes $(s_1, s_2, \dots, s_n)$. Tā kā valodai un izteiktajiem, uzrakstītajiem vārdiem dabiski piemīt secība, pielieto nosacīto varbūtību reizinājumu:

$$P(x) = \prod_{i=1}^{n} P(s_n | s_1, \dots, s_{n-1}). \tag{3.1}$$

Šāda pieeja ļauj izsekot un novērtēt $P(x)$, kā arī jebkurus nosacījumus formā $P(s_{n-k}, \dots, s_n | s_1, \dots, s_{n-k-1})$. Ir vērojami pamanāmi uzlabojumi to modeļu darbībā, kas var aprēķināt šīs nosacītās varbūtības – tādi ir, piemēram, pašuzmanības arhitektūru saturoši modeļi, tai skaitā iepriekš apskatītie *Transformer* modeļi.

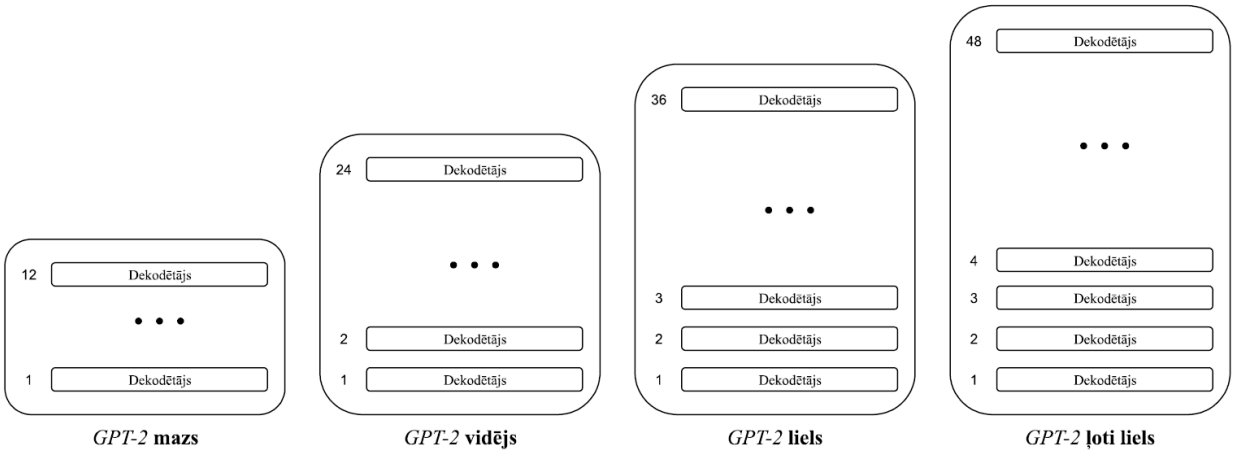
Mācīšanās veikt kādu konkrētu uzdevumu var tikt izteikta varbūtības sistēmā kā nosacītā varbūtība: $P(ievade|izvade)$. Tā kā vispārējai sistēmai ir jāspēj veikt daudz un dažādi uzdevumi pat ar vienu un to pašu ievadi, tai ir jābūt atkarīgai ne tikai no ievades, bet arī no veicamā uzdevuma. Tādā gadījumā nosacītā varbūtībā jāpievieno konkrētā uzdevuma definēšana: $P(ievade|izvade, konkrētais\ uzdevums)$.

Konkrēta uzdevuma veikšanai nosacījumi bieži vien tiek ieviesti jau arhitektūras līmenī, piemēram, kodētājos un dekodētājos vai arī ieejas un izejas slānī.

3.1. Modeļa arhitektūra

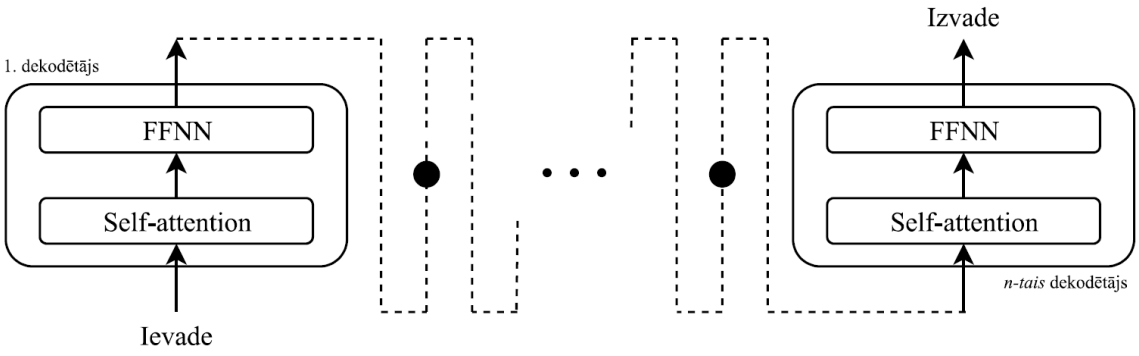
Kā jau no modeļa nosaukuma (*Generative Pretrained Transformer)* ir saprotams, tas ir veidots, izmantojot *Transformer* modeļa arhitektūru. Kodētāja-dekodētāja struktūra visbiežāk tiek izmantota tieši mašīntulkošanas uzdevumiem, bet *GPT-2* modeļa galvenais uzdevums ir spēt paredzēt nākamo vārdu tekstā, ņemot vērā visus iepriekšējos vārdus, tāpēc *GPT-2* modelī netiek izmantoti kodētāja bloki, bet gan tikai dekodētāja bloki. Jo vairāk modelī iekļauti dekodētāja bloki, jo ietilpīgāks un sarežģītāks ir pats modelis.

GPT-2 modelis pieejams četros izmēros – mazs (*small*), vidējs (*medium*), liels (*large*) un ļoti liels (*extra large*). Modeļi atšķiras gan ar dekodētāja bloku skaitu, gan parametru skaitu. Mazais modelis sastāv no 12 dekodētāja blokiem un 124 miljoniem parametru, vidējais no 24 blokiem un 355 miljoniem parametru, lielais no 36 blokiem un 774 miljoniem parametru, bet ļoti lielais – no 48 dekodētāja blokiem un 1558 miljoniem modeļa parametru (sk. 3.1. att.).



3.1. att. **GPT-2 modeļa izmēri**

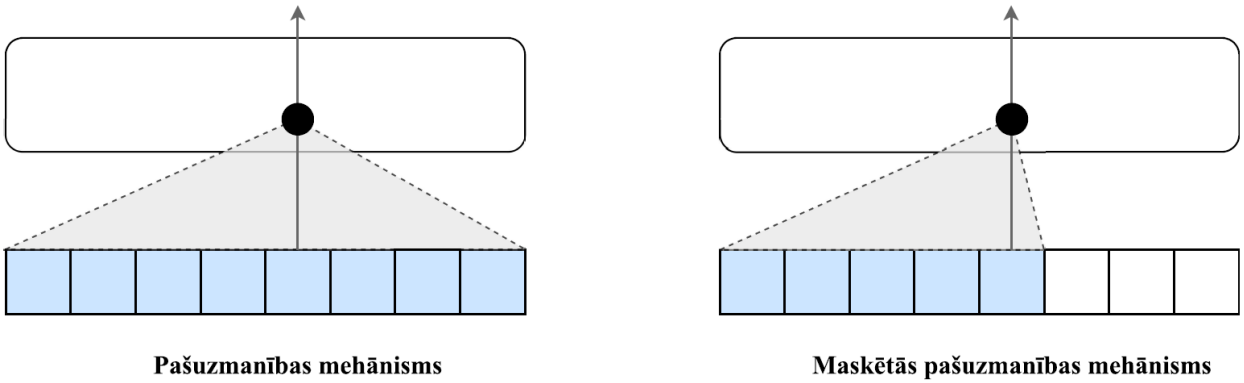
GPT-2 modelis sastāv no vairākiem secīgiem dekodētāja blokiem, un katram no tiem ir divi apakšslāņi – pašuzmanības slānis un FFNN slānis (sk. 3.2. att.).



3.2. att. **GPT-2 modeļa arhitektūra**

Standarta *Transformer* modeļa arhitektūrā dekodētājā tiek veikta vārdlietojuma kartēšana, kas savienota ar konteksta vektoru – tos abus ģenerē kodētājs, taču *GPT-2* modelī pirmā vārda kartēšanai konteksta vektoram ir nulles inicializācija.

Transformer modeļa arhitektūrā pašuzmanība tiek attiecināta uz visu apkārtējo kontekstu, piemēram, visi pārējie vārdi teikumā, taču *GPT-2* gadījumā tā vietā tiek izmantots maskētās pašuzmanības mehānisms – dekodētājam ir atļauts iegūt informāciju tikai no teikuma iepriekšējiem vārdiem un no paša apstrādājamā vārda. Tas notiek, izmantojot pārējo vārdu pozīciju aizsegšanu (sk. 3.3. att.). Pārējā ziņā *GPT-2* ir *Transformer* modeļa arhitektūras kopija. [18]



3.3. att. **Pašuzmanības un maskētās pašuzmanības mehānismi**

3.2. Modeļa darbība

GPT-2, tāpat kā tradicionālie valodu modeļi, vienlaikus izvada vienu tekstvienību. Pēc katras tekstvienības izvades tā tiek pievienota ievades sekvencei, pēcāk attiecīgi šī jaunā sekvence kļūst par modeļa ievadi tā nākamajā darbībā.

Vārdu vektori, kas tiek izmantoti *GPT-2* pirmajam slānim, nav vienkārši iegūti no dalīšanas tekstvienībās (angļu val. – tokenization), bet gan no baitu pāru kodēšanas. Baitu pāru kodēšanas shēma saspiež tekstvienībās sadalītu vārdu sarakstu noteiktā teksta korpusa lielumā, rekursīvi ievadot visbiežāk sastopamos vārdu komponentus unikālās vērtībās (piemēram, "ac" = 010010, "so" = 110010, "ta"=111000 , u.tml.). [19]

Novērtēšanas laikā modelis pārslēdzas uz paredzamo ievadi pa vienam vārdam. Tas tiek darīts, īslaicīgi saglabājot nepieciešamos pagātnes konteksta vektorus kā objekta rekvizītus. *GPT-2* ir apmācīts veikt standarta uzdevumu – paredzēt nākamo vārdu, ņemot vērā iepriekšējo vārdu secību.

Vienkāršākais veids, kā darbināt apmācītu *GPT-2* modeli, ir ļaut tam darboties pašam – veikt beznosacījumu paraugu (angļu val. – unconditional samples) ģenerēšanu, bet ir arī iespējams norādīt modelim sākuma tekstu, lai tas ģenerētu tekstu par noteiktu tematu – to sauc par interaktīvu nosacījumu paraugu (angļu val. – interactive conditional samples) ģenerēšanu. Beznosacījumu paraugu ģenerēšanas gadījumā apmācītajā modelī par ievadi tiek izmantota sākuma tekstvienība, ko definē kā <*endoftext*>. Ja ievadē ir šāda tekstvienība, modelis sāk jauna konteksta ģenerēšanu. Interaktīvu nosacījumu paraugu ģenerēšanas gadījumā modelim pirms teksta ģenerēšanas tiek sniegta kāda noteikta ievade, kuru modelis ņem vērā un pielāgo ģenerēto tekstu attiecīgajai tēmai. Abos modeļa darbības veidos tā darbības princips ir vienāds, jo abos gadījumos katra ievade ir viena tekstvienība – katra modeļa izvadītā tekstvienība tiek pievienota ievades sekvencei kā nākamā apstrādājamā tekstvienība, līdz ar to katras nākamās tekstvienības ģenerēšanai modelis ievadei izmanto visas iepriekšējos soļos sagatavotās tekstvienības.

Iepriekš apmācītam modelim ir savs valodas korpuss, kas tam ir zināms, un tajā katrai tekstvienībai, piemēram, vārdam, ir piešķirta noteikta varbūtība, ar kādu tam ir iespēja atrasties noteiktā tekstā blakus konkrētiem citiem vārdiem. Ir vārdi, kam varbūtība atrasties vienā tekstā ir

ļoti minimāla, piemēram, varbūtība vārdiem "futbols" un "saulespuķe" atrasties vienā tekstā, visticamāk, būs ļoti niecīga, taču varbūtība vārdiem "saule" un vasara" atrasties vienā tekstā varētu būt jau ievērojami lielāka. Tādējādi modelis, ģenerējot katru nākamo izvadi, var ņemt vērā ne tikai vienkārši ievades sekvenci, bet arī piemeklē nākamo vārdu, izvērtējot vārdu varbūtības atrasties blakus ievades vārdiem. Tas notiek, izmantojot apmācības laikā pielāgotos parametrus.

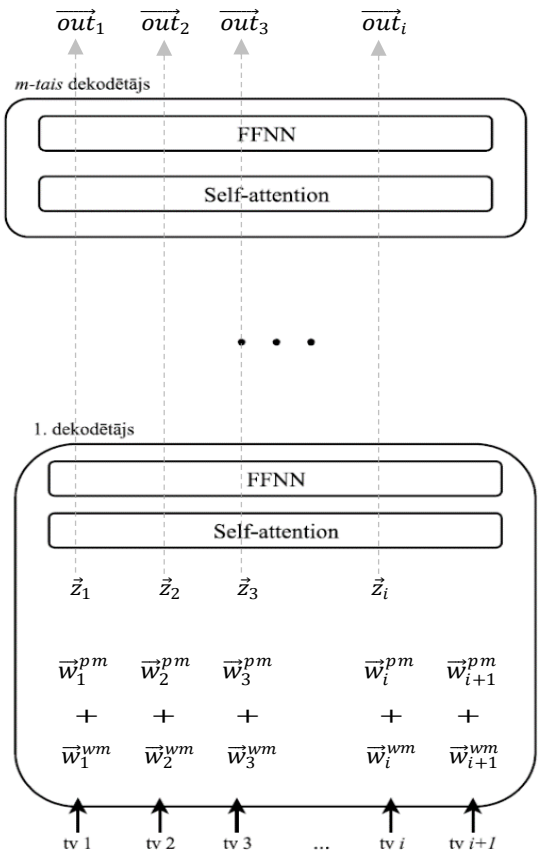
Lai arī katrai tekstvienībai piešķirtā varbūtība ir ļoti svarīga komponente modeļa darbībā, nevar teikt, ka vienmēr labākā un atbilstošākā nākamā tekstvienība būs tā, kurai ir noteikta augstākā varbūtība. Ja nākamā tekstvienība tiktu noteikta balstoties tikai uz lielāko varbūtību, pastāvētu ļoti liela iespēja, ka modelis izvadē sāktu bieži atkārtot vienu un to pašu tekstvienību – tas, protams, nav vēlams un nevar tikt uzskatīts par kvalitatīvu tekstu, jo, piemēram, dabiskā valodā pārāk bieži netiek atkārtoti vieni un tie paši vārdi. Lai varētu izvairīties no šādām situācijām, GPT-2 modelī ir parametrs *top-k*, ar kura palīdzību var noteikt to, no cik tekstvienībām, kam piešķirta augstākās varbūtības, modelis drīkst izvēlēties nākamo tekstvienību izvadei. Tiek norādīts, ka tieši teksta ģenerēšanas gadījumā kvalitatīvu tekstu iegūšanai labs vidusceļš ir parametram *top-k* izvēlēties vērtību 40. [18]

Nākamajā modeļa darbībā ievades sekvencei tiek pievienota pirmās darbības izvade un tiek veikta nākamās tekstvienības noteikšana jeb prognoze. Tā kā *GPT-2* modelis pēc katras iterācijas pievieno tikai vienu jaunu tekstvienību, būtu diezgan laikietilpīgi un ne pārāk efektīvi katru reizi atkārtoti interpretēt jau apstrādātās tekstvienības, tāpēc modelis aiztur jeb piefiksē katras tekstvienības aprēķinātos atslēgu un vērtību vektorus. Katrs pašuzmanības slānis piefiksē savu attiecīgo atslēgu un vērtību vektorus konkrētai tekstvienībai.

Arī *GPT-2* modeļa apmācības procesa gaitā tiek izstrādāta vektoros iestrādāto vārdu matrica W^{wm}, ko izmanto teksta paraugu ģenerēšanai. Konkrētās matricas rindās atrodas vektori katram valodas korpusā esošajam vārdam – tas ietver arī attiecīgā vārda semantisko nozīmi. Pieņemot, ka N ir valodas korpusā esošo tekstvienību skaits un D ir vārdu vektoru dimensija, matricas W^{wm} izmērs ir $N \times D$. Vārdu vektoru dimensija atšķiras dažādos *GPT-2* modeļos – mazā izmēra modelī vārdu vektoru dimensija ir 768, vidējā izmēra modelī tā ir 1024, lielā izmēra modelī tā ir 1280, bet ļoti lielā izmēra modelī šī dimensija ir 1600. Modeļa valodas korpusā esošo tekstvienību skaits ir 50257. [20]

Vēl viena matrica, kas tiek izstrādāta modeļa apmācības procesā, ir pozicionālā matrica W^{pm} – tā norāda ievades sekvences vārdu secību. Pieņemot, ka y ir modeļa konteksta vektora izmērs,

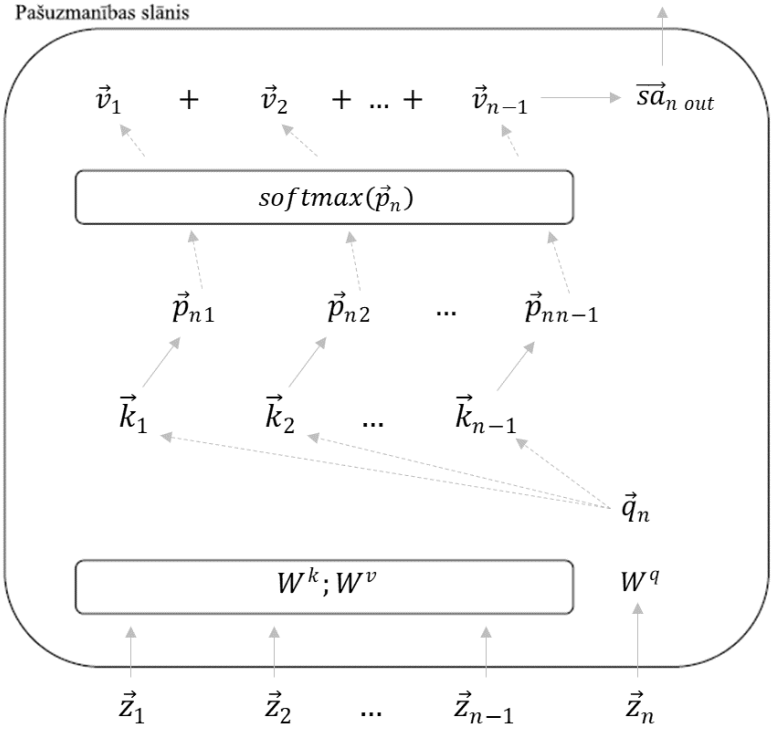
šīs matricas izmērs ir $y \times D$. Tieši *GPT-2* modelī šādas matricas izmērs ir $1024 \times D$, jo *GPT-2* modelī katra apstrādājamā tekstvienība var ieņemt pozīciju no 1 līdz 1024, tātad teksta ģenerēšanai tiek izmantotas 1024 tekstvienību pozīcijas. Pēc visu 1024 tekstvienību ģenerēšanas atrodas sākuma tekstvienība <*endoftext*>. Sākuma tekstvienība atrodas pozīcijā numur viens, līdz ar to nākamā tekstvienības pozīcija ir numur divi. Tālāk procesā modelis saņem ievades tekstvienību, tas atrod šai tekstvienībai atbilstošo vektoru no matricas W^{wm}, kā arī atbilstošo vektoru no matricas W^{pm}, lai pēc tam šos vektorus summētu. Šo darbību rezultātā iegūtais vektors kļūst par ievadi dekodētāja blokam. Šis vektors virzās pa visiem modeļa dekodētāja blokiem un tas tiek apstrādāts katrā dekodētāja blokā gan pašuzmanības apakšslānī, gan FFNN apakšslānī. Tā šis process turpinās līdz vektors ir nonācis līdz pēdējam modelī esošajam dekodētāja blokam. Kā jau iepriekš minēts, mazā izmēra *GPT-2* modelī ir 12 dekodētāja bloki, vidējā izmēra modelī 24, lielā izmēra modelī 36, bet ļoti lielā izmēra modelī – 48.



3.4. att. **Ievades vektoru virzība caur dekodētāja komponentēm**

Attēlā 3.4. redzams, ka katra tekstvienība (attēlā apzīmēta ar tv) modeļa dekodētāja blokos tiek ievadīta atsevišķi un katra modeļa izvade kļūst par jaunu tekstvienību raksturojošu vektoru. Šis jaunais vektors tiek pievienots ievades sekvencei, summēts ar pozīciju vektoru $\vec{w}_{i+1}^{pm}$, kas pēcāk kļūst par nākamo modeļa ievadi, tādējādi $\overrightarrow{out}_i = \vec{w}_{i+1}^{wm}$. Šis process nodrošina, ka katra nākamā modeļa ģenerētā tekstvienība tiek apstrādāta, ņemot vērā jau iepriekš esošos ievades datus.

Pašuzmanības slānī katrai tekstvienībai tiek piešķirts savs atslēgas vektors $\vec{k}_i$ un savs vērtības vektors $\vec{v}_i$. Tas palīdz ģenerēt nākamo tekstvienību tv_n, jo tās ģenerēšanai ir svarīgi zināt visas iepriekšējās tekstvienības tv_i, $i = 1, \dots, n-1$. Tekstvienība tv_n tiek apstrādāta visos dekodētāja blokos un tai apstrādes brīdī ir piešķirts vaicājuma vektors $\vec{q}_n$. [17]



3.5. att. **GPT-2 modeļa pašuzmanības mehānisms**

Kā redzams attēlā 3.5., pašuzmanības slānī modelis veic vaicājuma vektora $\vec{q}_n$ analīzi un visu iepriekšējo tekstvienību atslēgas vektoru $\vec{k}_i$ apskati. Tajā brīdī apstrādājamās tekstvienības vaicājuma vektors tiek kombinēts ar katru iepriekš esošo tekstvienību atslēgas vektoriem, lai piešķirtu katrai tekstvienībai novērtējumu p_i – tas nosaka, cik spēcīgi korelētas ir iepriekšējās

tekstvienības tv_i ar attiecīgā brīdī apstrādājamo tekstvienību tv_n. Pēc katras iepriekšējās tekstvienības novērtējuma p_{ni} ieguves katrai apstrādājamai tekstvienībai šos novērtējumus kopā var uztvert kā vektoru $\vec{p}_n$. Tā kā nepieciešams iegūt varbūtisku sadalījumu un normalizēt novērtējumu p_{ni} vērtības, arī *GPT-2* modelī tiek pielietota *softmax* funkcija. Ar tās palīdzību tiek iegūtas normalizētas vērtības $p_{ni\,norm}$, kā arī pēcāk vektors $\vec{p}_{n\,norm}$. Lai iegūtu dekodētāja bloka pašuzmanības slāņa izvadi, nepieciešams šos novērtējumus reizināt ar vērtību vektoriem un summēt: $\sum_{i=1}^{n} p_{ni}\vec{v}_i$. Šī izvade kļūst par FFNN slāņa ievadi, taču FFNN apakšslāņa izvade pēcāk kļūst par nākamā dekodētāja bloka pašuzmanības slāņa ievadi – tajā atkal turpinās tas pats process, taču katru reizi mainās apmācības laikā parametri.

Paša pēdējā dekodētāja bloka FFNN slāņa izvades vektors tiek reizināts ar matricu W^{wm}, tādējādi iegūstot varbūtiski sadalītu vektoru, kam dimensija ir N, jo N, kā jau iepriekš norādīts, ir valodas korpusā esošo vārdu skaits. Rezultātā iegūtā vektora lielākās vērtības ir tiem vārdiem, kuriem ir vislielākā varbūtība kļūt par modeļa izvadi, tomēr jāpatur prātā, ka tas ir arī atkarīgs no modeļa parametra *top_k*, jo, ja tas būs norādīts, piemēram, tikpat liels kā valodas korpusā esošo vārdu skaits, tas modelim ļaus izvēlēties no visiem valodas korpusā esošiem vārdiem, bet, ja, piemēram, parametrs $top\_k = 1$, tad modelim būs tikai viena izvēle, kuru vārdu izvēlēties – tas būs vārds ar visaugstāko varbūtību. [20]

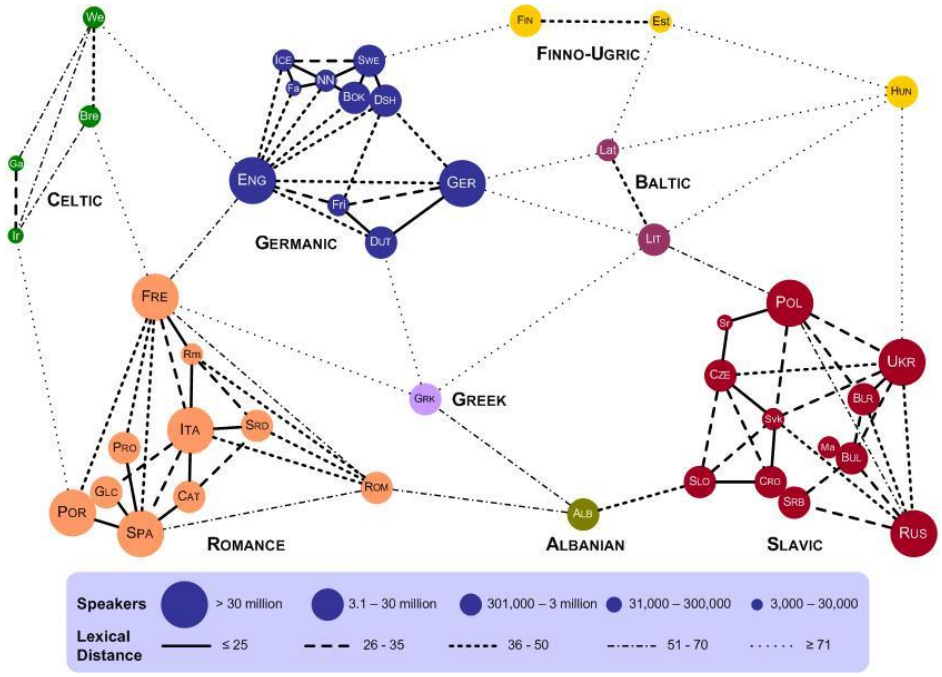
3.3. Modeļa pielāgošana citām valodām

Lielie ģeneratīvie valodu modeļi ir ļoti veiksmīgi angļu valodā, bet citas valodas atpaliek, visticamāk, datu un skaitļošanas ierobežojumu dēļ.

Kolorādospringsas Kolorādo universitātes (angļu val. – University of Colorado Colorado Springs) pētnieks Šauns Takers (angļu val. – Shaun Tucker) 2019. gada publikācijā ir aprakstījis eksperimentu, kur tika pielietots pielāgots (angļu val. – fine-tuned) *GPT-2* modelis sešām dažādām valodām – angļu, spāņu, bengāļu, hindi, asamiešu un ukraiņu valodai. Katrai no valodām tika veidots 1000 dzejoļu valodas korpuss un modelis tika trenēts gan uz visu valodas korpusu, gan uz izlases veida dzejoļu korpusiem, kas saturēja 200 un 500 dzejoļus. Galvenais secinājums, kas norādīts publikācijā, ir tāds, ka veiksmīgāk un arī laika ziņā ātrāk modelis tiek uztrenēts tām

valodām, kas ir tuvāk angļu valodai. [21]

Eiropas Parlamenta Terminoloģijas koordinācijas nodaļas (angļu val. – Terminology Coordination Unit of the European Parliament) publicētajā rakstā publicēta diagramma, kas parāda leksisko attālumu (vārdu krājuma atšķirības pakāpi) starp galvenajām Eiropas valodām. [22] Tajā redzams, ka leksiskais attālums starp angļu un latviešu valodu ir diezgan liels (sk. 3.6. att.).



3.6. att. **Leksiskais attālums starp galvenajām Eiropas valodām [22]**

Šauna Takera eksperimentā secināts, ka, trenējot modeli pat tikai uz 200, 500 vai 1000 dzejoļu lielu valodas korpusu, tas bija spējīgs ģenerēt dzejoļus. Protams, modeļa ģenerētie dzejoļi ir samērā tālu no cilvēka sacerētas dzejas, ko galvenokārt var pateikt pēc novērotajām gramatikas kļūdām. Kā jau iepriekš minēts, modelis uzrāda sliktākus rezultātus valodām, kam ir lielāks leksiskais attālums no angļu valodas – šo valodu ģenerētie piemēri saturēja vairāk gramatikas kļūdas. [21]

Pētnieks Afšins Khashei (angļu val. – Afshin Khashei) savā publikācijā raksta par dzejas ģenerēšanu persiešu valodā ar *GPT-2* modeli. Mērķis ir pielāgot jau iepriekš paša autora apmācīto *GPT-2* modeli, kas tika apmācīts uz liela persiešu valodas korpusa, jo vienkārši norādot sākuma

tekstu dzejolim un ievadot to *GPT-2* persiešu modelim, tas nespēja turpināt dzejoli ar atbilstošām atskaņām vai ritmu. Tas, visticamāk, ir tāpēc, ka persiešu *GPT-2* modelis tika apmācīts ar vienkārša teksta korpusu, līdz ar to modelis iemācījās attiecīgā valodas korpusa semantiskās likumsakarības un gramatiskos noteikumus. [23] [24]

Pētījuma autors pielietoja trīs metodes, lai panāktu rezultāta uzlabojumus. Pirmā metode ir *GPT-2* modeļa apmācība lielam vienkārša teksta korpusam (kā iepriekš minētajam *GPT-2* persiešu modelim), pēc tam kādu laiku turpinot apmācību ar klasisko persiešu dzeju. Otra metode ir *GPT-2* modeļa apmācība lielam teksta korpusam, kas ietver gan dzeju, gan tekstu, lielāku uzsvaru liekot tieši uz dzeju. Trešā metode ir *GPT-2* mazā izmēra modeļa apmācība tikai izmantojot klasiskās un modernās dzejas korpusu. Salīdzinot ar pirmo metodi, kur autors centās pielāgot iepriekš apmācīto persiešu *GPT-2* modeli, visas šīs trīs metodes uzlaboja rezultātu, modeļa ģenerētie piemēri kļuva vairāk poētiski, liriski, taču autors atzīst, ka tikai trešā metode spēja ģenerēt dzeju, kas būtu ritmiska un ar atskaņām. Tomēr arī trešajai metodei ir savi trūkumi – attiecīgajam modelim ir vāja semantiskā izpratne ārpus dzejas konteksta – ja sākuma rinda nav poētiska vai rakstīta par kādu konkrētu tēmu, kas dzejolī parasti neparādās, dzejoļa turpinājumam nebūs sasaistītas nozīmes. Tas, visticamāk, ir tāpēc, ka modelis tika apmācīts tikai uz dzejas korpusu, un dzejā ne vienmēr ir saprotama vārdu saistība savā starpā, jo tas tomēr ir radošs darbs.

Afšins Khashei norāda, ka pašlaik nav zināma tāda metrika, kas sistemātiski varētu izmērīt dažādus likumsakarības aspektus, kas varētu būt noderīgi, lai izvērtētu dzejas ģenerēšanu, piemēram, poētiskumu, radošumu, vārdu plūdumu, gramatikas likumu ievērošanu un tamlīdzīgi. Protams, ir modeļa tehniskā novērtējuma metrika (zuduma funkcija vai precizitāte), bet tā tikai norāda, cik labi modelis kopumā apgūst apmācības valodas korpusa likumsakarības. Teksta un īpaši dzejas rezultātu kvalitāti noteikti ietekmē visi iepriekš minētie likumsakarības aspekti, jo tie visbiežāk sniedz apmierinātību tieši no lasītāja viedokļa. Protams, katram cilvēkam ir dažāds skatījums uz radošiem darbiem, taču gramatikai vajadzētu būt ievērotai, kā arī visbiežāk dzejā svarīgs ir tieši vārdu plūdums.

Viens no autora ieteiktajiem veidiem, kā izmērīt ģeneratīvā modeļa kvalitāti, ir izmantot cilvēku atgriezenisko saiti – sniegt iespēju lasītājiem iepazīties ar modeļa ģenerēto dzeju un aicināt novērtēt to, vai attiecīgais dzejolis lasītājam patīk vai nepatīk. [23]

Autors citā publikācijā norāda, ka atsevišķiem pētniekiem vai mazām organizācijām nav iespējams kvalitatīvi apmācīt tādu valodu modeli kā *GPT-2* savai valodai, jo tam, pirmkārt, ir nepieciešama iespaidīga izmēra datu kopa un mazām valodām tādu būtu ļoti grūti apkopot, otrkārt,

ir nepieciešami ļoti lieli datora skaitļošanas resursi un, lai tādus iegūtu, ir nepieciešams liels finansējums. [24] Attiecīgi, nepieciešams mēģināt *GPT-2* modeli pielāgot uz jau apmācītu angļu valodas modeli, lai varētu ģenerēt tekstus savā valodā.

Pētnieks Ng Wai Fongs (angļu val. - Ng Wai Foong) ir izveidojis publikāciju, kurā parāda, ka *GPT-2* modeli var pielāgot, izmantojot valodas korpusu kādam konkrētam valodas ģenerēšanas uzdevumam. Autors iesaka izvēlēties modeli pielāgot kādam konkrētam uzdevumam angļu valodā, norādot, ka, pielāgojot modeli citai valodai, pastāv iespēja nesasniegt pārāk labus rezultātus pat tad, ja ir visi nepieciešamie datora skaitļošanas resursi, laiks un valodas korpuss. Iespējams, lai to parādītu, autors publikācijā izmanto valodas korpusu japāņu valodā, lai modelis varētu ģenerēt tekstus japāņu valodā. Tā kā attiecīgā publikācija ir vairāk kā pamācība, kā var veikt *GPT-2* modeļa pielāgošanu, autors neveic nekādus gala secinājumus, tikai ir iespējams redzēt, ka modelis pa soļiem veic pielāgošanu un izvada ģenerētos piemērus. Sākotnēji tie nav lieliski piemēri, jo dzejoļos atrodami gan angļu valodas, gan japāņu valodas vārdi, kā arī veidojas dažādi japāņu valodas vārdi, kas, pēc autora norādītā, nav sakarīgi un gramatiski korekti. Pētnieks norāda, ka nepieciešams turpināt modeļa pielāgošanas procesu un palielināt iterāciju skaitu, lai iegūtu kvalitatīvākus piemērus. [25]

Pētnieks Aršabi Kayal (angļu val. – Arshabhi Kayal) izsaka atšķirīgu viedokli iepriekš apskatītajiem – tā kā *GPT-2* tika izlaists angļu valodā, tas apgrūtina teksta ģenerēšanu citā valodā, kas nav angļu valoda. Tāpēc autors iesaka apmācīt savu *GPT-2* modeli nepieciešamajā valodā teksta ģenerēšanai. Publikācijā autors izvēlas modeli apmācīt bengāļu valodā. Autors norāda, ka attiecīgā publikācija sniedz ieskatu, kā ir iespējams apmācīt *GPT-2* modeli jebkurā valodā, norādot, ka tas nav līdzvērtīgs dažiem pieejamajiem iepriekš apmācītajiem modeļiem, piemēram, oriģinālajam *OpenAI GPT-2* modelim, jo, lai sasniegtu tādus modeļa rezultātus, ir nepieciešams daudz vairāk apmācības datu un datora skaitļošanas resursu. [26]

4. DZEJAS ĢENERĒŠANA LATVIEŠU VALODĀ, IZMANTOJOT VALODAS MODELI

Maģistra darba praktiskā daļa paredzēta dzejas ģenerēšanai latviešu valodā, izmantojot valodas modeli. Kā jau iepriekš minēts, ir iespējams izmantot iepriekš apmācītu valodas modeli un to pielāgot nepieciešamajam uzdevumam ar tam specializētu valodas korpusu, kā arī no jauna apmācīt modeli uz attiecīgo valodas korpusu.

4.1. Valodas korpusa sagatavošana

Viena no svarīgākajām komponentēm valodas ģenerēšanas modeļa apmācīšanai ir laba, apjomīga un kvalitatīva valodas korpusa sagatavošana, ar kuru tiks veikta modeļa pielāgošana vai apmācība.

Tā kā mērķis ir ģenerēt dzeju latviešu valodā, tad šajā gadījumā valodas korpuss jāveido no dažādiem dzejoļiem, tautasdziesmām vai līdzīgiem pantiem. Lielākais datu apjoms tika iegūts, pateicoties sabiedrībai ar ierobežotu atbildību "Tilde", kas, noslēdzot abpusēju apliecinājumu, piekrita nodot dzejas datus latviešu valodā pētījumu veikšanai šī maģistra darba ietvaros. Saņemti tika dažādu latviešu, kā arī tulkoti ārzemju dzejnieku un dramaturgu darbi – Māra Čaklā, Klāva Elsberga, Raiņa un Knuta Skujenieka dzeja, Imanta Ziedoņa poēma, librets rokoperai pēc Andreja Pumpura eposa "Lāčplēsis", Viljama Šekspīra lugu "Romeo un Džuljeta", "Hamlets" un "Karalis Līrs" tulkojumi latviešu valodā, Henriksa Ibsena lugas "Pērs Gints" tulkojums latviešu valodā, Johana Volfganga Gētes lugas "Fausts" tulkojums latviešu valodā. Pēc datu apskates un analīzes, tika secināts, ka visas iepriekš minētās lugas neatbilst dzejas ritmiem un atskaņām, tāpēc tās netika izmantotas modelī izmantotajā valodas korpusā. Papildus tika atlasīti Jāņa Ziemeļnieka, Eduarda Veidenbauma, Austras Skujiņas, Jāņa Poruka, Ausekļa, Aspazijas un Jura Alunāna dzejoļi no tīmekļa vietnes *www.letonika.lv* sadaļas "Lirika jeb dzeja" [27], kā arī latviešu tautasdziesmas, kas apkopotas un pieejamas Latvijas Universitātes Matemātikas un informātikas institūta Mākslīgā intelekta laboratorijas tīmekļa vietnē *www.valoda.ailab.lv*. [28]

Visi iegūtie dzejoļi un tautasdziesmas tika apvienotas vienā teksta failā, pirms tam tos apstrādājot, izmantojot programmēšanas valodu *Python*:

```
string = open('Caklais_dzeja.txt').read()
new_str = re.sub('[^a-zA-Z\n\āĀčČēĒģĢīĪķĶļĻņŅšŠūŪžŽ\,.]', ' ', string)
open('Caklais_dzeja_apstradats.txt', 'w').write(new_str)
```

Tā tika apstrādāts katrs dokuments, lai izdzēstu tipiskos simbolus, kas atrodami dzejoļu krājumos – pārsvarā tas bija simbols "*".

Veikta visu sagatavoto dokumentu apvienošana vienā teksta failā, ko izmantot tālāk par kopējo valodas korpusu:

```
filenames = ['Caklais_dzeja_apstradats.txt',
             'Knuta_Skujenieka_dzejas_izlase_apstradats.txt',
             'Iesien_balta_lakatina_apstradats.txt',
             'Elsbergs_Velci_teeti_apstradats.txt',
             'Elsbergs_Beeda_uz_Nebeedu_apstradats.txt',
              'Caklais_dzeja2_apstradats.txt',
              'Caklais_dzeja_apstradats.txt',
              'Raina_arpuskrajumu_dzejoli_apstradats.txt',
              'Rainis_papildus_apstradats.txt']
with open('Dzejolu_korpuss.txt', 'w') as outfile:
    for fname in filenames:
        with open(fname) as infile:
            outfile.write(infile.read())
```

Datu apvienošanas rezultātā iegūts valodas korpuss, kas tiks izmantots valodas modeļa pielāgošanai un apmācībai – tajā kopā ir 287029 vārdi.

4.2. Modeļa GPT-2 pielāgošana

Lai veiktu *GPT-2* pielāgošanu dzejas ģenerēšanai latviešu valodā, darbā izmantota jau iepriekšējā nodaļā pieminētā pētnieka Ng Wai Fonga publikācija un ieteiktā metode. Oriģinālais pirmkods ir pieejams tīmekļa vietnē *GitHub*: *https://github.com/nshepperd/gpt-2*.

Visas darbā izmantotās modeļu versijas trenētas, izmantojot *Google Colaboratory* (saīsinājumā "*Colab*"). Tas ir *Google Research* produkts. Izmantojot *Colab*, ikvienam ir iespēja rakstīt un izpildīt *Python* kodu, izmantojot pārlūkprogrammu – tas ir mitināts *Jupyter* piezīmju

bloka (angļu val. – notebook) pakalpojums, kura lietošanai nav nepieciešama iestatīšana, un tas nodrošina bezmaksas piekļuvi skaitļošanas resursiem – gan grafiskajam procesoram (GPU), gan tenzorprocesoram (TPU). [29]

Pirmkārt nepieciešams klonēt *GitHub* repozitoriju aktīvajā direktorijā:

```
!git clone https://github.com/nshepperd/gpt-2.git
```

Veic *Python* moduļu instalēšanu, izmantojot nodrošināto prasību failu *requirements.txt*. Prasību fails satur nepieciešamos moduļus, to versijas, kas ir nepieciešami koda izpildei:

```
 !pip3 install -r "/content/gpt-2/requirements.txt"
```

Tālāk nepieciešams lejupielādēt bāzes modeli, norādot modeļa izmēru. Komandu uzvednē jānorāda to mapi, kurā ir fails *download_model.py*:

```
#Mazais modelis
!python3 /content/gpt-2/download_model.py 117M
#Vidējais modelis
#!python3 download_model.py 345M
#Lielais modelis
#!python3 download_model.py 774M
```

Jāatceras, ka nepieciešams lietot *UTF-8* kodējumu, lai latviešu valodas burti tiktu uztverti korekti:

```
!export PYTHONIOENCODING = UTF-8
```

Tiek veikta modeļa trenēšana, norādot sagatavotā valodas korpusa teksta faila nosaukumu un modeļa nosaukumu/izmēru:

```
!PYTHONPATH = src /content/gpt-2/train.py
--dataset /content/Dzejolu_korpuss.txt
--model_name '117M'
```

Pēc noklusējuma modelis ģenerē teksta paraugus ik pēc 100 treniņa soļiem, bet, ja ir nepieciešams, paraugu ģenerēšanas biežumu var mainīt. Pēc katra soļa modelis izvada zaudējuma funkcijas vērtību attiecīgajā solī un vidējo zaudējuma funkcijas vērtību:

```
Training...
[1 | 38.30] loss=4.11 avg=4.11
[2 | 68.99] loss=4.75 avg=4.43
[3 | 99.65] loss=4.51 avg=4.46
```

```
[4 | 130.31] loss=3.62 avg=4.24
[5 | 160.97] loss=4.26 avg=4.25
[6 | 191.62] loss=5.09 avg=4.39
[7 | 222.25] loss=3.68 avg=4.29
[8 | 253.02] loss=4.44 avg=4.31
[9 | 283.68] loss=4.42 avg=4.32
[10 | 314.33] loss=4.23 avg=4.31
          ...
[95 | 2986.77] loss=3.74 avg=3.91
[96 | 3018.28] loss=3.31 avg=3.90
[97 | 3049.77] loss=3.92 avg=3.90
[98 | 3081.34] loss=4.40 avg=3.91
[99 | 3112.79] loss=3.02 avg=3.90
Generating samples...
======== SAMPLE 1 ========
Indonesia's president on Wednesday named an alternate president to replace Prime Minister Binali Yildirim as the country looks for a pro-democracy, pro-Western leader.


KARGUN, Malaysia – The Indonesian president has ordered a two-person transition committee to be led by Yildirima. However, a senior official in the government says, in order to save a political prisoner, they need the votes of at least a third party or a minority.


"In order to make sure that a pro-democracy, pro-Western candidate is elected, a political committee, it's at a stage where it is clear which candidate is a sure bet for a pro-democracy," said deputy premier Dusko Markau in a telephone interview.
```

Pēc pirmajiem 100 treniņa soļiem mazākais GPT-2 modelis izvada paraugu principā tikai angļu valodā, bet, turpinot modeļa trenēšanu, var redzēt, ka modelis mācās un izvades kvalitāte uzlabojas. Piemēram, pēc 300 treniņa soļiem izvade vairs nesastāv tikai no angļu valodas vārdiem:

```
[298 | 9538.69] loss=3.39 avg=3.50
[299 | 9570.02] loss=3.66 avg=3.50
Generating samples...
======== SAMPLE 1 ========
klik tik tik, otra man ik kara bij viju,
Kan sakašamā jau vienu sūrāsās krīt.
Kar vietas vienās vietās zeltaš drejā,
```

```
Kad virts par sauli vienās ir sakašamā jau,
Kad tik izla dienās, nei mazdaši,

Tad kāri apātām uz sauli vienās zeltaš drejā ik līdz uzelinātām nau.

Kad vietas vienās vietās zeltaš drejā,

Tad lūk, vienās par izkiņa čeikām sauli, pietājēm kad.
Lai uz lūk, vienās un viena, un zemas un nezinās pūst,
Ko at
Rīt tas    ko sauli viena, kas tu roku,
Viss tas
Ko kur ču   un dienas
Un man at kas bijsārī, tešiksiks
I, no ļātām, tešiksiks ādības
Ka sakašu sauli jau zeltaš, mēdzinā,
Ka at, kas viņa zeltām, dviņšiksības.
Un viņšiksiks   nau saulis
Un zu kalas kār sakar krīt, naulām nau saulis.
Un izmiņas turpēt, ko tu dvaist mājās,
Lai man parek, dzēviņi skalēt   nau saulis.
```

Lai arī pēc pirmajiem 300 treniņa soļiem modelis neizvada tikai angļu valodas vārdus, ir skaidri redzams, ka tam ir grūtības tik strauji iemācīties no pievienotā valodas korpusa – izvades teksts nav latviešu valodā, tā uzbūve nav līdzīga dzejoļu uzbūvei, kā arī zaudējuma funkcijas vērtība vēl joprojām ir liela, tāpēc modeļa trenēšana jāturpina.

Pēc 400 treniņa soļiem modelis sāk ģenerēt tekstus, kas pēc uzbūves jau nedaudz vairāk līdzinās dzejai – vārdu skaits rindās ir mazāks, kā arī parādās latviešu valodas vārdi:

```
[398 | 12768.23] loss=3.03 avg=3.35
[399 | 12799.69] loss=3.86 avg=3.35
Generating samples...

======== SAMPLE 1 ========
PAPA PASSA PĒREĪT
Krļiem sika vienīgs nāk,
```

```
Cēdu no tev nāk.
Kapaulas tīs nauvies
PĒĒSA PĒREĪT
Papās pēcis jauto
Nāk, par saule pēmēt
Kā saule nevarēs,
Manādāšās par cīduš,
Vienu stāva
Un jau sāp katris rāvi
Dvēs izpokāt lauzies.
Uglež un plēvot, no šem tec,
Un jau jau tālām, un jau jau tālām.
Kā cienu, karvērts krūs nauk,
Es māžos, manas rāts, vienai sirdes
Un saulūs, manas kuršās.
Šināsi sarkā, un tāk cēdu ziedī,
Tāvērus nāk īst, no tāk cēdu naudi.
```

Arī pēc 500 treniņa soļiem rezultāts, šķiet, uzlabojas – rindas kļūst īsākas un atkal parādās latviešu valodas vārdi:

```
[498 | 15995.95] loss=3.20 avg=3.26
[499 | 16027.47] loss=2.81 avg=3.26

Generating samples...
======== SAMPLE 1 ========
Ne ērzim, ko
Ko tu vien
Tiem tik mani.
  Ko neemē, dzilām,
Rūstķim, kusiem
Iņa mani.
  Viļās, nelis
Vedam mājīs, saviņas
Kad sirds sāpes mājas
Ak, kad manākus,
```

```
Mājas vēlo.
  Sēkles mana vēl
Mākē, tik arāt sauli
Mājas skarām tāpēm
Daiļa smilienies un viņiem

Tik tung iztā izšām
Tik ētļā kā māk tai dzīve
Tam, tik ētļā man neļūsu
Pirms, kas iet manā

Pēcu dzeni tevi ēto
Tumlēcēt man kākām
Tei, dzimmes, kad mana vien.
Mūs savās, putes.
```

Modeļa trenēšana turpinājās, bet tā notika ļoti lēni un *Google Colab* rīks katru reizi pārtrauca sesijas darbību, nedodot iespēju saglabāt modeli un atspējojot iespēju izmantot GPU vai TPU uz aptuveni 12, citreiz pat 24 stundām. Jo biežāk tika izmantots *Google Colab* rīks sintakses aktivizēšanai un modeļa trenēšanai, jo īsāks bija atļautais sesijas darbības laiks. *Google Research* savā tīmekļa vietnē raksta: "Lietotājiem, kuri izmanto *Colab* ilgstošiem aprēķiniem, vai lietotājiem, kuri nesen ir izmantojuši vairāk *Colab* resursu, pastāv lielāka iespējamība, ka tiks ierobežoti lietošanas ierobežojumi un viņu piekļuve GPU un TPU uz laiku tiks ierobežota. Lietotājus, kurus interesē augstāki un stabilāki lietošanas ierobežojumi, var interesēt *Colab Pro* un *Pro+*." [29] Diemžēl turpat rakstīts, ka *Colab Pro* vai *Pro+* ir pieejams tikai noteiktās valstīs, kuru sarakstā nav iekļauta Latvija.

Kad pēc ierobežošanas laika beigām atkal parādījās iespēja izmantot GPU, tika mēģināts trenēt vidējā izmēra *GPT-2* modeli. Pat, ja netika pārtraukta sesija un ierobežota piekļuve GPU, neilgu laiku pēc trenēšanas komandas palaišanas izvadē parādījās ziņojums "^C", process apstājās un parādījās uznirstošais logs, kas ziņo, ka resursi ir pārsniegti un šo darbību nav iespējams veikt. Par šādu paziņojumu neizdevās atrast oficiālu *Google Research* informāciju, bet, mēģinot vēl vairākas reizes, neizdevās trenēt vidējā izmēra *GPT-2* modeli.

Diemžēl šo pieeju neizdevās attīstīt tālāk un trenēt modeli pietiekami ilgi, lai iegūtu kvalitatīvus dzejoļu paraugus latviešu valodā.

4.3. Valodas modeļa apmācība

Lai veiktu valodas modeļa apmācību, izmantota jau iepriekšējā nodaļa pieminētā Aršabi Kayal publikācija un testēta autora pieeja valodas modeļa apmācībai, lai ģenerētu tekstu citā valodā.

Arī šajā gadījumā koda rakstīšanai un izpildīšanai izmantots *Google Colab*.

Pirms darba sākšanas nepieciešams instalēt vajadzīgās bibliotēkas, šajā gadījumā:

- *TensorFlow;*
- *Tokenizers;*
- *Transformers;*
- *Pathlib.*

Bibliotēka *TensorFlow* ir speciāli paredzēta augstas veiktspējas skaitlisko aprēķinu veikšanai, izmantojot centrālo procesoru (CPU), GPU vai TPU pieejamos resursus. Bibliotēka Pathlib sniedz iespēju veiksmīgāk un ērtāk pārvaldīt sistēmas ceļu, tādā veidā ļaujot piekļūt dažādiem failiem. Bibliotēka Transformers nodrošina pieeju dažādiem iepriekš sagatavotiem modeļiem NLP uzdevumu veikšanai. Šī bibliotēka ir savietojama gan ar TensorFlow, gan ar PyTorch bibliotēku – šajā gadījumā izmantota ir TensorFlow bibliotēka. Visbeidzot, bibliotēka Tokenizers ir paredzēta apmācības datu sadalīšanai tekstvienībās.

Modeļa apmācības veikšanai ar sagatavoto valodas korpusu sākotnēji nepieciešams veikt apmācības datu dalīšanu tekstvienībās:

```
import os
from tokenizers.models import BPE
from tokenizers import Tokenizer


from tokenizers.normalizers import NFKC, Sequence
from tokenizers.decoders import ByteLevel as ByteLevelDecoder
from tokenizers.trainers import BpeTrainer
from tokenizers.pre_tokenizers import ByteLevel
```

```
class BPE_token(object):
    def __init__(self):
        self.tokenizer = Tokenizer(BPE())
        self.tokenizer.normalizer = Sequence([
            NFKC()
        ])
        self.tokenizer.pre_tokenizer = ByteLevel()
        self.tokenizer.decoder = ByteLevelDecoder()

    def bpe_train(self, paths):

        trainer = BpeTrainer(vocab_size = 50000,
                  show_progress = True,
                  initial alphabet = ByteLevel.alphabet(),
                  special_tokens = [
            "<s>",
            "<pad>",
            "</s>",

            "<unk>",
            "<mask>"
        ])
        self.tokenizer.train(paths, trainer)

    def save_tokenizer(self, location, prefix = None):
        if not os.path.exists(location):
            os.makedirs(location)
        self.tokenizer.model.save(location, prefix)
```

Arī šajā gadījumā, tāpat kā GPT-2 modelī, izmanto baitu pāru kodēšanu no bibliotēkas *Tokenizer*. Tā nodrošina to, lai dažādas vārdu formas netiktu uzskatītas par atšķirīgām. Turklāt baitu pāru kodēšana nav tik smalka kā rakstzīmju līmeņa kodējums, kas nesaglabā nekādu konkrēta vārda vērtību. Tā kā teksts ir latviešu valodā, ir svarīgi veikt unikoda normalizēšanu – to veic, izmantojot NFC un *ByteLevel* apstrādi.

Visiem valodas korpusa failiem ir jāatrodas vienā mapē, jo no tās tiek atlasīti visi teksta faili un, izmantojot tikko definēto klasi *BPE_token*, tiek veikta datu sadalīšana tekstvienībās un to saglabāšana norādītajā direktorijā, kas šajā gadījumā ir mape "tokenized_data":

```
from tokenizers import BPE_token
from pathlib import Path
import os

paths = [str(x) for x in Path("./dzejolu_korpuss/").glob("**/*.txt")]
tokenizer = BPE_token()
tokenizer.bpe_train(paths)

save_path = 'tokenized_data'
tokenizer.save_tokenizer(save_path)
```

Kad teksta dati ir sadalīti tekstvienībās, tiek veidots modelis ar tikko izveidoto datu un bibliotēkas *Transformers* palīdzību – tajā ir pieejamas *GPT-2* modeļa definēšanas un konfigurēšanas iespējas:

```
import tensorflow as tf
from transformers import GPT2Config, TFGPT2LMHeadModel, GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained(save_path)
tokenizer.add_special_tokens({
  "eos_token": "</s>",
  "bos_token": "<s>",
  "unk_token": "<unk>",
  "pad_token": "<pad>",
  "mask_token": "<mask>"
})

config = GPT2Config(
  vocab_size = tokenizer.vocab_size,
  bos_token_id = tokenizer.bos_token_id,
  eos_token_id = tokenizer.eos_token_id
)
```

```
model = TFGPT2LMHeadModel(config)
```

GPT2Tokenizer tiek izmantots, lai definētu speciālās tekstvienības, kas nepieciešamas modeļa darbībai. Ar *eos_token* definē konteksta beigu tekstvienību, ar *bos_token* definē konteksta sākuma tekstvienību, ar *unk_token* definē nezināmu tekstvienību, ko izmanto gadījumā, kad iegūtais izvades vektors ar vislielāko varbūtību būt nākamajai tekstvienībai ir tāds vektors, kas nav atrodams modeļa valodas korpusā.

GPT2Config tiek izmantots modeļa konfigurācijai, kur vārdu krājums modelī ir atbilstošs dalīšanas tekstvienībās procesā iegūtajam – dekodētāja bloku skaits ir atbilstošs *GPT-2* mazā izmēra modelim, tātad 12 dekodētāja bloki, konteksta vektora dimensija, tāpat kā *GPT-2* modelim pēc noklusējuma ir 1024 un vārdu vektoru dimensija arī ir atbilstoša *GPT-2* mazā izmēra modelim, tātad 768.

TFGPT2LMHeadModel tiek izmantots kā definētā konfigurācija modelī – tas ir viens no bibliotēkā *Transformer* iekļautajiem neironu tīklu ģeneratoriem, kas ir pielāgots tieši *GPT-2* modelim.

No apmācībai paredzētā valodas korpusa mapes tiek nolasīti visi faili (ja tādi ir vairāki), katram nolasītajam failam beigās pievienojot konteksta beigu tekstvienību. No nolasītā teksta izveido vienu teksta rindu, ko sadala tekstvienībās un šo rindu tālāk izmanto modeļa apmācīšanai nepieciešamās izvades ģenerēšanai:

```
single_string = ''
for filename in paths:
  with open(filename, "r", encoding = 'utf-8') as f:
   x = f.read()
  single_string += x + tokenizer.eos_token
string_tokenized = tokenizer.encode(single_string)
```

Rezultātā viss tekstvienībās sadalītais valodas korpuss tiek sadalīts vienāda izmēra intervālos, kur katrā no tiem ir simts tekstvienības, un tas tiek saglabāts vektora formā, līdz ar to veidojas datu kopa, ar kuru tiek apmācīts modelis:

```
examples = []
block_size = 100
BATCH_SIZE = 12
BUFFER_SIZE = 1000
```

```
for i in range(0, len(string_tokenized) - block_size + 1, block_size):
  examples.append(string_tokenized[i:i + block_size])
inputs, labels = [], []

for ex in examples:
  inputs.append(ex[:-1])
  labels.append(ex[1:])

dataset = tf.data.Dataset.from_tensor_slices((inputs, labels))
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=T
rue)
```

Nākamajā solī definē modeļa veidošanai nepieciešamās funckijas – tās ir optimizācijas, zaudējuma un metrikas funkcijas. Šim nolūkam tiek izmantota *Tensorflow Keras* saskarne, kas ir populāra dažādu mašīnmācīšanās uzdevumu veikšanai, izmantojot *Python*.

Optimizācijas funkciju veido ar *Adam* algoritmu, kas izmanto stohastiskā gradienta optimizācijas metodi. Adam algoritms tiek izmantots, lai paātrinātu gradienta nolaišanās algoritmu, ņemot vērā gradientu eksponenciāli svērto vidējo vērtību. Izmantojot vidējos rādītājus, algoritms ātrāk tuvojas minimumam. Šī metode atzīta par patiešām efektīvu, strādājot ar lielu problēmu, kas saistīta ar daudz datu vai parametru. Tas prasa arī mazāk atmiņas resursu. [30] Zaudējuma funkciju veido, izmantojot *SparseCategoricalCrossentropy* klasi, jo šajā gadījumā tekstvienības ir skaitlisku vektoru formā. Metrikas funkciju veido, izmantojot *SparseCategoricalAccuracy* klasi. Šī metrika izveido divus lokālos mainīgos — kopējo un skaitu, kas tiek izmantoti, lai aprēķinātu biežumu, ar kādu paredzētās vērtības atbilst patiesajām vērtībām. Aprēķinātais biežums tiek atgriezts kā bināra precizitāte.

```
optimizer = tf.keras.optimizers.Adam(learning_rate = 3e-5,
                                     epsilon = 1e-08,
                                     clipnorm = 1.0)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True)

metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model.compile(optimizer = optimizer,
```

```
                    loss = [loss, *[None] * model.config.n_layer],
                    metrics = [metric])
```

Nākamajā solī nepieciešams definēt apmācību etapu (angļu val. – epoch) skaitu, lai uzsāktu pielāgotā modeļa apmācīšanu dzejas ģenerēšanai latviešu valodā:

```
num_epoch = 60
history = model.fit(dataset, epochs=num_epoch)
```

Pēc katra apmācību etapa modelis izvada iepriekš definēto funkciju vērtības.

Kad apmācību process ir beidzies, modeli saglabā atsevišķā mapē, lai to pēc tam varētu ielādēt no jauna noteiktā direktorijā un izmantot dzejas paraugu ģenerēšanai latviešu valodā:

```
from transformers import WEIGHTS_NAME, CONFIG_NAME
import os
output_dir = './modelis-apmacits/'

if not os.path.exists(output_dir):
  os.mkdir(output_dir)
model_to_save = model.module if hasattr(model, 'module') else model
output_model_file = os.path.join(output_dir, WEIGHTS_NAME)
output_config_file = os.path.join(output_dir, CONFIG_NAME)

model.save_pretrained(output_dir)
model_to_save.config.to_json_file(output_config_file)

tokenizer.save_pretrained(output_dir)
```

Modeli un ar to saistītos failus nepieciešams dublēt citā direktorijā, lai brīdī, kad *Google Colab* pārtrauc savienojumu ar izpildlaika vidi, paciet̄gi trenētais modelis un visi iegūtie dati netiktu pazaudēti:

```
import shutil
shutil.copy('./modelis-apmacits/config.json',
            './drive/MyDrive/modelis-apmacits/')
shutil.copy('./modelis-apmacits/merges.txt',
            './drive/MyDrive/modelis-apmacits/')
shutil.copy('./modelis-apmacits/special_tokens_map.json',
            './drive/MyDrive/modelis-apmacits/')
```

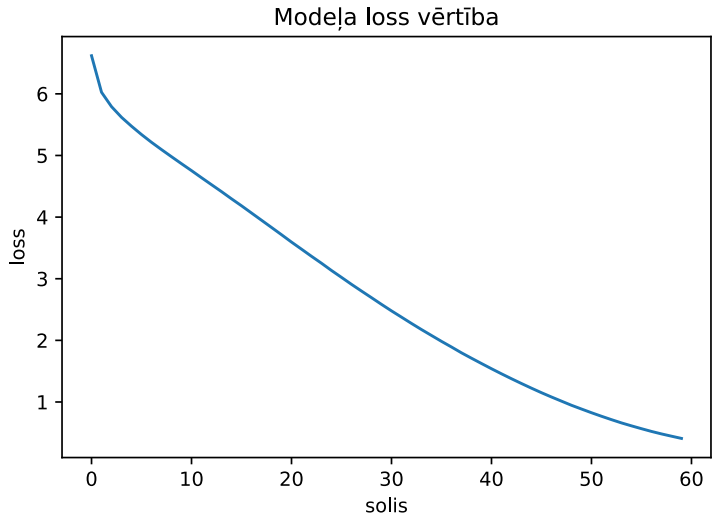
```
shutil.copy('./modelis-apmacits/tf_model.h5',
            './drive/MyDrive/modelis-apmacits/')
shutil.copy('./modelis-apmacits/tokenizer_config.json',
            './drive/MyDrive/modelis-apmacits/')
shutil.copy('./modelis-apmacits/vocab.json',
            './drive/MyDrive/modelis-apmacits/')
```

Pēc modeļa apmācības ir iespējams grafiski attēlot zaudējuma funkcijas un precizitātes vērtības katrā no etapiem (šajā gadījumā 60):

```
import matplotlib.pyplot as plt


plt.plot(history.history['loss'])
plt.title('Modeļa loss vērtība')
plt.ylabel('loss')
plt.xlabel('solis')
plt.savefig("modelis-loss-vertiba.svg")
plt.show()
```

Iegūto grafiku skatīt attēlā 4.1. Redzams, ka zaudējuma funkcijas vērtība ar katru etapu samazinās, kas arī bija sagaidāms.



4.1. att. **Apmācītā modeļa zaudējuma funkcijas vērtība atkarībā no etapu skaita**

4.4. Dzejoļu paraugu ģenerēšana latviešu valodā ar apmācīto valodas modeli

Lai ģenerētu paraugus dzejai latviešu valodā, nepieciešams aktīvajā direktorijā ielādēt iepriekš apmācīto valodas modeli un definēt tā parametrus:

```
import tensorflow as tf
from transformers import GPT2Config, TFGPT2LMHeadModel, GPT2Tokenizer
modelis = './drive/MyDrive/modelis-apmacits/'
tokenizer = GPT2Tokenizer.from_pretrained(modelis)
model = TFGPT2LMHeadModel.from_pretrained(modelis)
start_text = " "
input_ids = tokenizer.encode(start_text, return_tensors = 'tf')
model_output = model.generate(
 input_ids,
 max_length = 75,
 do_sample = True,
 num_beams = 3,
 top_k = 40,
 top_p = 0.95,
 temperature = 0.8,
 no_repeat_ngram_size = 4,
 num_return_sequences = 3)

print(tokenizer.decode(model_output[0], skip_special_tokens = True))
```

Ar *start_text* palīdzību tiek norādīta sākuma ievade modelim – tā tiek sadalīta tekstvienībās un izmantota modeļa darbināšanai kā sākuma ievade.

Ar *input_ids* palīdzību tiek norādīta iepriekš tekstvienībās sadalīta modeļa sākuma ievade.

Ar *max_length* palīdzību var norādīt un mainīt ģenerējamā teksta parauga garumu.

Ar *num_beams* palīdzību var norādīt, cik sekojošu vārdu varbūtības tiek ņemtas vērā, veicot teksta ģenerēšanu. Kā jau iepriekš apskatīts, lai ģenerētu nākamos vārdus, modelis ņem vērā vārdu varbūtības, tādējādi var būt tāda situācija, ka pirmais vārds ir ar mazāku varbūtību, bet vārds, kas seko pēc tā, ir ar augstāku varbūtību – šādā situācijā pirmais vārds tiek pazaudēts. Ar šī parametra

palīdzību ir iespējams novērst šādas situācijas, jo, ģenerējot teksta turpinājumu, tiks izvēlēti parametra vērtībā norādīti sekojoši vārdi tā, lai to varbūtību reizinājums būtu augstākais.

Ar *num_return_sequences* palīdzību var norādīt, cik vārdus nepieciešams ģenerēt no to vārdu skaita, kuru varbūtību reizinājumi ir ņemti vērā, kad modelis izvēlas nākamo vārdu secību, tātad šis parametrs ir saistīts ar iepriekš aprakstīto *num_beams* parametru. Tādējādi parametra *num_return_sequences* vērtība nedrīkst būt lielāka par *num_beams* parametra vērtību.

Ar *no_repeat_ngram_size* palīdzību var kontrolēt to, lai modelim nebūtu tendence ģenerēt atkārtojošu tekstu – ja kāda noteikta vārdu secība jau ir bijusi tekstā, tad, ģenerējot turpmāko tekstu, tādai vārdu secībai varbūtība tiek piešķirta nulle, līdz ar to tāda pati vārdu secība tekstā vairs neparādīsies. Šis parametrs īpaši noderīgs varētu būt dzejas ģenerēšanai – protams, dzejoļos mēdz atkārtoties vārdu savienojumi, taču, ja tas notiek pārāk bieži, visticamāk, tas neuzrunās lasītāju.

Ar *temperature* palīdzību var panākt izteiktāku vārdu varbūtību sadalījumu – palielinās augstākas iespējamības vārdu varbūtības, bet zemākas iespējamības vārdu varbūtības samazinās.

Kā jau iepriekš apskatīts, ar *top_k* palīdzību var norādīt, no cik vārdiem ar vislielākajām varbūtībām izvēlēties nākamo vārdu, tādējādi, iespējams, novēršot situāciju, kad modelis bieži ģenerē vienu un to pašu vārdu vai frāzi.

Šajā pieejā modeļa trenēšanas procesā netiek izvadīti paraugi, līdz ar to visi turpmāk norādītie paraugi ir ģenerēti pēc modeļa apmācības veikšanas:

1. paraugs, norādot modelim sākuma ievadi "*Zvaigzne spīd*":

Zvaigzne spīd debešos
Mēness smaida joprojām.
Tikko bij tas riets
Nakts plaši staigā

2. paraugs, norādot modelim sākuma ievadi "*Mēness logā*":

Mēness logā cauri spīd
Citu dienām spīd,
Zemesvētku acis spīd
Vārdi saulē spīd

Līdz saule spīd
Ilgi, es redzu,
Tur augšā tik maigi.
Svētku spulgo gaismā,
Līdz ar smaršu dvēsli,
Vai tās saule gaida,
Mēness reiz saule
Zemazgā baltā zeltā.

3. paraugs, norādot modelim sākuma ievadi "*Jauna diena*":

Jauna diena jūs iegūt,
Skan prom ar mīlu.
Kad zinu es zinu, ka tām cita cita reiz,
Kas manu mīlu reiz ātrāk reiz ziedēt,
Kā balta sāpes un mani minu
Kad roka pasaule man cita cita
Tad, es zinu, kas man laimes mīlu
Es zinu, kā cita cita.
Man šķiet, ka tu guvi,
Bet tu jau laimes mīlā diezgan
Tad zini, kur es.

4. paraugs, norādot modelim sākuma ievadi "*Mežs*":

Mežs un putni
Un visa tāle
Mākoņu zaros
Pāri vārtiem
Ar jaunu valsti
Prom no bailēm
Vēl top, vēl skumst

Vēl jūt vienas sāpes

5. paraugs, norādot modelim sākuma ievadi "*Skumjas māc*":

Skumjas māc dūmi klīst.
Tās pērkons pērkons klīst un plūst,
Līdz pat atpakaļ spīd
Aiz pilsētas kalniem
Šķīst acīs slīd
Kā vilnis zibens klīst

6. paraugs, norādot modelim sākuma ievadi "*Saule spīd*":

Saule spīd un visus silda,
Mīla brīve visam cauri spīd.
Vieglas, skaidras visas sirdis,
Lielas, jautras domas.
Laikam, telpai pāri ceļ,
Sniegt un cerēt jaunu zemi.
Visiem līdzīgiem būt un labiem
Lai top katrs labs par sevi.

7. paraugs, norādot modelim sākuma ievadi "*Tu smejies*":

Tu smejies, met ar roku,
Tad klusi mājā uz dusu.
Ko tu neproti draugs,
Tā jau zini, ka tā pati dvēsle plauks.
Es viņu leju katru brīdi
Ak lai tik skaisti mirdz.

8. paraugs, norādot modelim sākuma ievadi "*Smaids*":

Smaids, kad vakars zied,
Es dziedu par Tevi,
Jo nav ko raudāt
Nav ko skumt.
Nakts jau aiz loga nāk,
Laime kā lietus līst.
Es tūlīt jau nāku,
Tu dari savādāku.

9. paraugs, norādot modelim sākuma ievadi "*Vētra nāk*":

Vētra nāk un veļas,
Lietus arī ceļas.
Zūd Visas domas, krusa veļas
Zūd saikne ar ikdienību dūmi ceļas

10. paraugs, norādot modelim sākuma ievadi "*Esmu laimīgs*":

Esmu laimīgs un auksts kā zelta stari.
To savādi, ko jūti,
tas ir vecs.
Kā slaidais ugunī pūš,
uzzina cilvēka koks.
Ja zāle tas ir te.
Dienas ceļiem, bet nekas.
Kaut arī par vēlu.
Rītdien ir jāpeld,
pāri, protams,
ja vien gribi.

Veicot šo un citu modeļa ģenerēto paraugu analīzi, ir redzams, ka modelis spēj ģenerēt tekstus, kas ir dzejoļu formātā – ar salīdzinoši īsām rindām. Dažos dzejoļos iespējams novērot

tēmas nostiprināšanu, izmantojot līdzīgus vārdus vai vārdus, kas saistās ar norādīto sākuma ievades vārdu vai vārdu savienojumu. Jāatzīmē, ka modelis ne pārāk labi veic atskaņu ģenerēšanu, tikai dažviet ir vērojamas atskaņas, biežāk redzams, ka modelis ģenerē divu secīgu rindu beigās vienādus vārdus. Arī no gramatikas viedokļa dzejoļi ne vienmēr ir korekti – gan vārdu locījuma ziņā, gan pareizrakstības ziņā. Tas, visticamāk, ir stipri atkarīgs no valodas korpusa, kuru modelis izmanto apmācības gaitā. Noteikti būtu vērtīgi sīkāk izskatīt valodas korpusā iekļautos dzejoļus, koriģēt tos, lai tur būtu sastopami mazāk vecvārdi, apvidvārdi un tamlīdzīgi. Domājams, ka arī palielinot valodas korpusa apjomu un tajā iekļauto dzejoļu skaitu modelis uzrādītu vēl labākus rezultātus.

4.5. Ģenerēto dzejoļu paraugu novērtējums

Kā jau iepriekš minēts, Afšins Khashei savā publikācijā norāda, ka pašlaik nav zināma tāda metrika, kas sistemātiski varētu izmērīt dažādus likumsakarības aspektus, kas varētu būt noderīgi, lai izvērtētu dzejas ģenerēšanu, piemēram, poētiskumu, radošumu, vārdu plūdumu, gramatikas likumu ievērošanu un tamlīdzīgi. Viens no autora ieteiktajiem veidiem, kā novērtēt modeļa ģenerētos paraugus, ir izmantot cilvēku atgriezenisko saiti, tādēļ tika nolemts veikt aptauju.

Sākotnēji aptaujā bija paredzēts iekļaut desmit modeļa ģenerētos paraugus un lūgt respondentiem tos novērtēt, taču tad nebūtu datu, ar ko salīdzināt iegūtos rezultātus, tāpēc aptaujā iekļauti seši modeļa ģenerētie paraugi, kas nedaudz koriģēti, un četri latviešu dzejnieku dzejoļi. Aptaujā iekļautie dzejoļi apskatāmi 4.1. tabulā.

4.1. tabula

Aptaujā iekļautie dzejoļi

Kārtas numurs	Dzejolis	Dzejoļa autors
1.	Smaids, kad vakars zied, Es dziedu par Tevi, Jo nav ko raudāt, Nav ko skumt. Nakts jau aiz loga nāk, Laime kā lietus līst. Es tūlīt jau nāku, Tu mani dari savādāku.	Apmācītais modelis

4.1. tabulas turpinājums

Kārtas numurs	Dzejolis	Dzejoļa autors
2.	Smiekli bez gala Ritē un trīsē, Līdz agram rītam Līksmība laistās. Dzīvības prieka Pārpilnas sirdis Pārplūst no katra Niecīga vārdiņa. Atbild uz katru Smiekliem bez gala, Izsauc no katra Smieklus bez gala.	Rainis
3.	Tu smejies, met ar roku, Tad klusi mājā ej uz dusu. Lūk, ko tu neproti, draugs, Tā jau zini, ka tā pati dvēsle plauks. Es viņu leju katru brīdi Ak, lai tik skaisti mirdz.	Apmācītais modelis
4.	Vētra nāk un veļas, Lietus arī ceļas. Zūd Visas domas, krusa veļas Zūd saikne ar ikdienību un dūmi ceļas.	Apmācītais modelis
5.	Vairāk par cilvēkiem pašiem Es cilvēci mīlējis esmu, Viņai man likās aizvien Lielāks un vienīgais svars. Cilvēki maksāja man Ar to pašu - tie gāja man garām, Kodols tiem likās mans darbs, Atmesta čaumala - es.	Rainis
6.	Zvaigzne spīd debešos, Mēness smaida joprojām. Tikko bija tas riets, Tagad nakts plaši staigā.	Apmācītais modelis
7.	Mūžam dziņa un mūžam šaubas. Mūžam meklēt un nerast nekad Gars un galva, un miesas, un sirds Paši sevi saēd un sakvēlojas Un pelnus iznēsā vējš.	Rainis
8.	Laiks pēdas dzēš — bet pirmveids palicis, Un dzīves koks gan atkal uzplaucis, Bet jaunās lapas saista pērnais zibens. Pat jūru nes uz pleciem drūmais dibens, Kam pāri viļņo smaidošs līmenis, Un laipnās gaismas bargais tēvs bij — zibens.	Aspazija

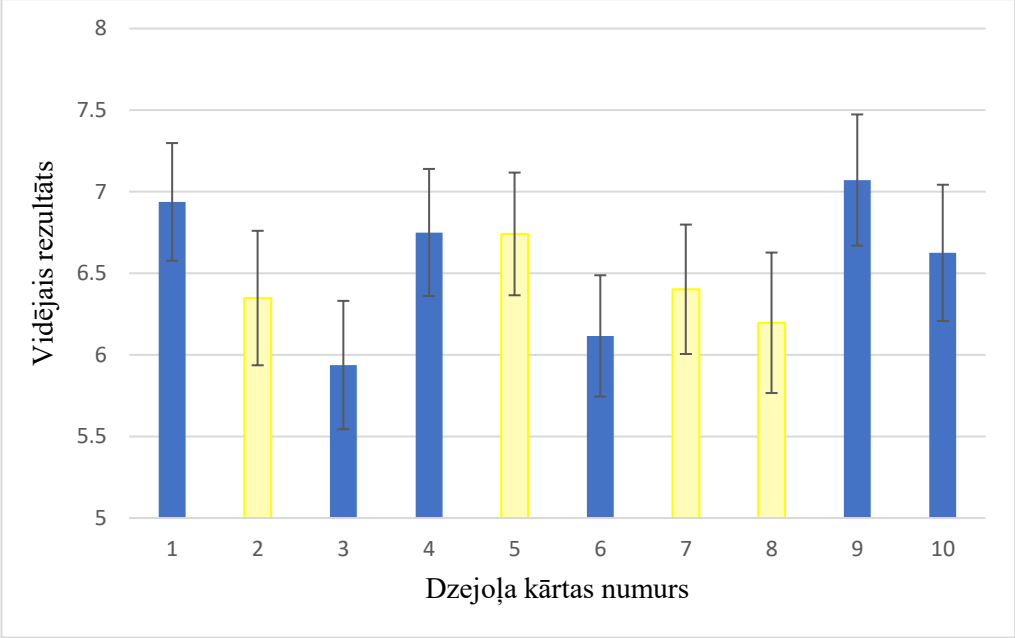
4.1. tabulas turpinājums

Kārtas numurs	Dzejolis	Dzejoļa autors
9.	Saule spīd un visus silda, Mīla, brīve visam cauri spīd. Vieglas, skaidras visas sirdis, Lielas, jautras visas domas. Laikam, telpai pāri ceļas, Sniegt un cerēt jaunu zemi. Visiem līdzīgiem būt un labiem Lai top katrs labs par sevi.	Apmācītais modelis
10.	Mežs un putni Un visa tāle Mākoņu zaros Pāri vārtiem Ar jaunu valsti Prom no bailēm Vēl top, vēl skumst Vēl jūt vienas sāpes	Apmācītais modelis

Respondentu uzdevums bija novērtēt katru dzejoli no 1 līdz 10, kur vērtējums 1 norāda, ka dzejolis nemaz nepatika/neuzrunāja, bet vērtējums 10 norāda, ka dzejolis ļoti patika/uzrunāja.

Aptaujā piedalījās 112 respondenti, no kuriem 87 bija sievietes un 25 vīrieši. Respondentu vidējais vecums ir 39 gadi.

Aptauja tika veikta tiešsaistē, izmantojot tīmekļa vietni *www.visidati.lv*, aptaujas jautājumi un tās forma ir dota 1. pielikumā, aptaujas automātiski ģenerētais pārskats par respondentiem un vidējiem vērtējumiem dots 2. pielikumā, bet aptaujā iegūtie individuālie respondentu vērtējumi doti 3. pielikumā. Aptaujā iegūtais vidējais vērtējums grafiski apskatāms 4.3. attēlā, kur dzejoļa kārtas numurs atbilst aptaujas, kā arī 4.1. tabulas secībai, un zilā krāsa izmantota dzejoļiem, kas ģenerēti, izmantojot apmācīto valodas modeli, dzeltenā krāsa – dzejoļiem, kuru autors ir kāds latviešu dzejnieks.



4.2. att. **Aptaujā iegūtais dzejoļu vidējais vērtējums un to ticamības intervāli**

Kā redzams 4.2. attēlā, ir grūti viennozīmīgi novērtēt iegūtos rezultātus. Četri no sešiem modeļa ģenerētiem dzejoļiem ir spējuši uzrādīt labus rezultātus, pat labākus kā latviešu autoru ģenerētie dzejoļi. Divi modeļa ģenerētie dzejoļi nav uzrunājuši respondentus, iespējams tāpēc, ka tie ir samērā īsi un tajos nav novērojamas atskaņas.

Ņemot vērā aprēķinātos un 4.2. attēlā redzamos ticamības intervālus, iespējams secināt, ka atšķirība starp lielāko daļu no dzejoļiem nav statistiski nozīmīga – ja salīdzina modeļa ģenerētos piemērus ar latviešu dzejnieka daiļradi, tad tikai starp astoto un devīto dzejoli atšķirība ir statistiski nozīmīga. Ticamības intervālu, kā arī standartnovirzes un standartkļūdas aprēķini doti 4. pielikumā.

Ja apkopo aptaujas rezultātus sešiem modeļa ģenerētiem dzejoļiem, tad to vidējais iegūtais vērtējums ir 6.58, savukārt četriem rakstnieku radītiem dzejoļiem vidējais iegūtais vērtējums ir 6.42.

REZULTĀTI

Ir izpētīta datora ģenerētas dzejas vēsture, apskatītas dažādas dabiskās valodas apstrādes pieejas, izzinot to attīstību, kas sākusies jau pirms vairākiem gadu desmitiem.

Padziļināti izpētīts gan *Transformer* modelis, gan *GPT-2* modelis, kā rezultātā izdevās pielietot divas metodes dzejas ģenerēšanai latviešu valodā.

Apskatītas dažādas publikācijas, kur citu valstu pētnieki dalās pieredzē par valodas modeļu pielāgošanu vai apmācību citām valodām, kas nav angļu valoda. Iegūti secinājumi par to, ka izveidot tik labu valodas modeli, kā tas ir angļu valodā, ir sarežģīti, jo, pirmkārt, ne visās valodās ir pieejams tik liels datu apjoms valodas korpusa sagatavošanai, īpaši tas attiecas uz mazāk izplatītām valodām, tai skaitā latviešu valodu, otrkārt, fiziskai personai vai nelieliem uzņēmumiem pietrūkst skaitļošanas resursu vai finanses, lai varētu veikt tik apjomīgu modeļa apmācību vai pielāgošanu.

Lai veiktu dzejas ģenerēšanu latviešu valodā, iegūti dati no dažādiem avotiem un sagatavots pielāgots valodas korpuss attiecīgajam uzdevumam.

Oriģinālā priekšapmācītā *GPT-2* modeļa pielāgošanu neizdevās sekmīgi veikt līdz galam ar izmantoto metodi *Google Research* veidotā *Google Colab* rīka resursu ierobežojumu dēļ.

Sekmīgi izdevās veikt modeļa apmācību uz sagatavoto valodas korpusu, izmantojot oriģinālā GPT-2 modeļa iestrādnes. Iegūto dzejas paraugu izvērtēšana nav vienkārša, jo tai nav noteiktas metrikas, kā to kvalitāti varētu pareizāk izmērīt, ar šo problēmu saskaras arī citu valstu kolēģi.

Veikta aptauja, izmantojot tiešsaistes vietni, pēc kuras iegūtajiem rezultātiem var secināt, ka arī ar samērā mazu un vidēji apstrādātu valodas korpusu var iegūt cilvēkus uzrunājošus dzejoļus. Tas, protams, ir atkarīgs no katra cilvēka individuālām sajūtām – noteikti būs cilvēki, kam neviens no darbā iegūtajiem modeļa ģenerētajiem dzejoļiem nešķitīs saistošs. Ja salīdzina modeļa ģenerētos dzejoļus ar dažiem rakstnieku radītiem dzejoļiem, tos var nosaukt pat par savstarpēji konkurējošiem.

Aptaujā tika iekļauti tie dzejoļi, kas vislabāk spēja pieturēties pie vienas tēmas un tie, kam parādījās pāris atskaņas. Ne visi dzejoļi, kas tika ģenerēti ar valodas modeli, izdevās tik veiksmīgi – liela daļa dzejoļu ir ar gramatikas kļūdām, ne pārāk labi uztverami, sasaista dzejolī diametrāli

pretējas lietas vai arī pārāk bieži atkārto vienu un to pašu vārdu. Īpaši tas novērojams rindu beigās. Šādas ģenerēto dzejoļu nepilnības var būt latviešu valodas īpatnību dēļ. Piemēram, ja salīdzina latviešu valodu ar angļu valodu, var secināt, ka latviešu valodā bieži tiek lietoti dažādi priedēkļi, piedēkļi, kā arī tiek veikta darbības vārdu locīšana – angļu valodā liela daļa teikumu tiek formulēti vienkāršāk. Gramatikas kļūdas ģenerētajos modeļa dzejoļos noteikti lielā mērā ir atkarīgas no valodas korpusa, ko izmanto modeļa apmācībai – tas ir nepieciešams ļoti kvalitatīvs, lai panāktu labu rezultātu.

SECINĀJUMI

Dabiskā valoda ir piepildīta ar dažādām valodas īpatnībām, kas ievērojami apgrūtina tāda valodas modeļa izveidi, kas spēj precīzi noteikt teksta vai balss datu patieso nozīmi. Dabiskās valodas ģenerēšana sevī ietver vairākus sarežģītus uzdevumus. Aizvien biežāk šo uzdevumu risināšanai tiek izvēlēti modeļi, kas balstās uz *Transformer* modeļa arhitektūru. Šie modeļi ir iepriekš apmācīti ar lielām valodu datu kopām, lai pēc tam tos varētu tikai nedaudz pielāgot specifiskākiem uzdevumiem. Šie modeļi pašlaik ļoti veiksmīgi attīstās un ir vieni no zināmākajiem valodas modeļiem.

Darbā izvirzītais mērķis ir sasniegts – ir izdevies ar valodas modeli ģenerēt dzeju latviešu valodā, kā arī novērtēt tā darbību, veicot tiešsaistes aptauju. Daļa no datora ģenerētajiem dzejoļiem uzrāda labus rezultātus pēc aptaujas respondentu domām.

Lielā daļā no ģenerētajiem dzejoļiem latviešu valodā ir gramatikas kļūdas, vārdu atkārtošanās un modelis visbiežāk nespēj izveidot atskaņas pa rindām. Lai to uzlabotu, būtu nepieciešams izveidot daudz apjomīgāku un noteikti arī kvalitatīvāku valodas korpusu, kā arī atrast lielus skaitļošanas resursus, lai varētu veikt modeļa apmācību.

Ņemot vērā, ka darbā izmantotais modelis ir apmācīts uz ievērojami mazāka valodas korpusa kā oriģinālais *GPT-2* modelis angļu valodā, kā arī iztrūka pieejas lielākiem skaitļošanas resursiem, noteikti būtu iespējams ģenerēt ļoti kvalitatīvus dzejoļus latviešu valodā, kuriem nebūtu iepriekš minēto nepilnību.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] M. D'Ambrosio, "The Early Computer Poetry and Concrete Poetry", 2018 [atsauce 09.12.2021]. Pieejams: The Early Computer Poetry and Concrete Poetry | PDF | Poetry | Haiku (scribd.com)

[2] IBM Cloud Education, "Natural Language Processing (NLP)", 02.07.2020 [atsauce 10.12.2021]. Pieejams: https://www.ibm.com/cloud/learn/natural-language-processing

[3] Y. Goldberg, *Neural Network Methods for Natural Language Processing*. Toronto, Canada: Morgan & Claypool, 2017.

[4] NLPEA, "History of NLP" [atsauce 10.12.2021]. Pieejams: https://nlpea.com/international-nlp-association-of-excellence-nlpea/history-nlp

[5] Great Learning Team, "Understanding Curse of Dimensionality", 01.10.2020 [atsauce 20.12.2021]. Pieejams: https://www.mygreatlearning.com/blog/understanding-curse-of-dimensionality/

[6] A.Vaswani et al., "Attention Is All You Need," in *31st Conference on Neural Information Processing Systems*, 2017 [atsauce 05.01.2022]. Pieejams: https://arxiv.org/pdf/1706.03762.pdf

[7] GeeksforGeeks, "Introduction to Recurrent Neural Network", 03.10.2018 [atsauce 07.01.2022]. Pieejams: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

[8] G. Giacaglia, "How Transformers Work", 11.03.2019 [atsauce 20.01.2022]. Pieejams: https://towardsdatascience.com/transformers-141e32e69591

[9] H. Kortschak, "Attention and Transformer Models", 16.11.2020 [atsauce 18.01.2022]. Pieejams: https://towardsdatascience.com/attention-and-transformer-models-fe667f958378

[10] J.Brownlee, "What Are Word Embeddings for Text?", 11.10.2017 [atsauce 21.01.2022]. Pieejams: https://machinelearningmastery.com/what-are-word-embeddings/

[11] J. Alammar, "The Illustrated Transformer", 27.06.2018 [atsauce 22.01.2022]. Pieejams: https://jalammar.github.io/illustrated-transformer/

[12] 360DigiTMG Team, “GPT vs BERT”, 23.07.2021 [atsauce 21.01.2022]. Pieejams: https://360digitmg.com/gpt-vs-bert#bidirectional-bert-and-autoregressive-gpt

[13] Hugging Face, “Summary of the models” [atsauce 22.01.2022]. Pieejams: https://huggingface.co/docs/transformers/model_summary

[14] A. Hemanthika, “Natural Language Inferencing (NLI) Task: Demonstration Using Kaggle Dataset”, 19.05.2021 [atsauce 22.01.2022]. Pieejams: https://affine.ai/natural-language-inferencing-nli-task-demonstration-using-kaggle-dataset/

[15] B. Ammu, “GPT-3: All you need to know about the AI language model”, 17.12.2021 [atsauce 23.01.2022]. Pieejams: https://www.sigmoid.com/blogs/gpt-3-all-you-need-to-know-about-the-ai-language-model/

[16] A. Radford et al., “Better Language Models and Their Implications”, 14.02.2019 [atsauce 22.12.2021]. Pieejams: https://openai.com/blog/better-language-models/

[17] E.Voita, “Sequence to Sequence (seq2seq) and Attention”, 30.04.2022 [atsauce 05.05.2022]. Pieejams: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

[18] J. Alammar, “The Illustrated GPT-2 (Visualizing Transformer Language Models)”, 12.08.2019 [atsauce 13.04.2022]. Pieejams: https://jalammar.github.io/illustrated-gpt2/

[19] A. Bilogur, “Notes on GPT-2 and BERT models”, 23.01.2020 [atsauce 13.04.2022]. Pieejams: https://www.kaggle.com/code/residentmario/notes-on-gpt-2-and-bert-models/

[20] Hugging Face, “OpenAI GPT2” [atsauce 17.04.2022]. Pieejams: https://huggingface.co/transformers/v3.0.2/model_doc/gpt2.html

[21] S. Tucker, *Transfer Learning Across Languages with Poetry*, 2019 [atsauce 17.04.2022]. Pieejams: https://www.proquest.com/openview/194737039beaa87888761587df84a20c/1?pq-origsite=gscholar&cbl=18750&diss=y

[22] Terminology Coordination Unit of the European Parliament, “Lexical Distance Among the Languages of Europe”, 14.01.2014 [atsauce 18.04.2022]. Pieejams: https://termcoord.eu/2014/01/lexical-distance-languages-europe/

[23] A. Khashei, “Writing Persian Poetry with GPT-2.0”, 29.03.2020 [atsauce 30.03.2022].

Pieejams: https://khashei.medium.com/writing-persian-poetry-with-gpt-2-0-71b7197317ea

[24] A. Khashei, "A Not-so-Dangerous AI in the Persian Language", 09.03.2020 [atsauce 30.03.2022]. Pieejams: https://khashei.medium.com/a-not-so-dangerous-ai-in-the-persian-language-39172a641c84

[25] N.W. Foong, "Beginner's Guide to Retrain GPT-2 (117M) to Generate Custom Text Content", 13.05.2019 [atsauce 28.03.2022]. Pieejams: https://medium.com/ai-innovation/beginners-guide-to-retrain-gpt-2-117m-to-generate-custom-text-content-8bb5363d8b7f

[26] A. Kayal, "Text Generation in any language with GPT-2", 01.09.2020 [atsauce 28.03.2022]. Pieejams: https://medium.com/engineered-publicis-sapient/text-generation-in-any-language-with-gpt-2-e8fba8656167

[27] Letonika.lv, sadaļa "Lirika jeb dzeja" [atsauce 14.03.2022]. Pieejams: https://www.letonika.lv/literatura/Section.aspx?f=1&id=2189831

[28] Latviešu valodas tautasdziesmu korpuss [atsauce 14.03.2022]. Pieejams: http://valoda.ailab.lv/latval/vispareji/tautasdz/t00.htm

[29] Google Research, "Frequently Asked Questions" [atsauce 18.04.2022]. Pieejams: https://research.google.com/colaboratory/faq.html

[30] GeeksforGeeks, "Intuition of Adam Optimizer", 24.10.2020 [atsauce 01.05.2022]. Pieejams: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/

PIELIKUMI

Veiktās aptaujas jautājumi

Datora ģenerētas dzejas novērtēšana

Sveiki!
Esmu Latvijas Universitātes Datorikas fakultātes studente. Maģistra darba izstrādes ietvaros veicu aptauju, kas palīdzēs novērtēt datora ģenerētu dzeju.

Lūdzu izlasiet dzejoli un skalā no 1 līdz 10 novērtējiet, vai attiecīgais dzejolis Jūs uzrunāja/Jums patika, vai tas Jūs neuzrunāja/Jums nepatika.
1 - Nemaz nepatika/neuzrunāja
10 - Ļoti patika/uzrunāja

Aptaujas aizpildīšana aizņems ne vairāk kā 5 minūtes Jūsu laika.
Aptaujas rezultāti būs konfidenciāli un tiks izmantoti tikai apkopotā veidā.
Liels paldies jau iepriekš!

Norādiet informāciju par sevi:

Dzimums ----
Vecums ----

1. dzejolis:

Smaids, kad vakars zied,
Es dziedu par Tevi,
Jo nav ko raudāt,
Nav ko skumt.
Nakts jau aiz loga nāk,
Laime kā lietus līst.
Es tūlīt jau nāku,
Tu mani dari savādāku.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

	1	2	3	4	5	6	7	8	9	10	
Nemaz nepatika/neuzrunāja	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Ļoti patika/uzrunāja

2. dzejolis:

Smiekli bez gala
Ritē un trīsē,
Līdz agram rītam
Līksmība laistās.
Dzīvības prieka
Pārpilnas sirdis
Pārplūst no katra
Niecīga vārdiņa.
Atbild uz katru
Smiekliem bez gala,
Izsauc no katra
Smieklus bez gala.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

	1	2	3	4	5	6	7	8	9	10	
Nemaz nepatika/neuzrunāja	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Ļoti patika/uzrunāja

3. dzejolis:

Tu smejies, met ar roku,
Tad klusi mājā ej uz dusu.
Lūk, ko tu neproti, draugs,
Tā jau zini, ka tā pati dvēsle plauks.
Es viņu leju katru brīdi
Ak, lai tik skaisti mirdz.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

	1	2	3	4	5	6	7	8	9	10	
Nemaz nepatika/neuzrunāja	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Ļoti patika/uzrunāja

4. dzejolis:

Vētra nāk un veļas,
Lietus arī ceļas.
Zūd Visas domas, krusa veļas,
Zūd saikne ar ikdienību un dūmi ceļas.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

	1	2	3	4	5	6	7	8	9	10	
Nemaz nepatika/neuzrunāja	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Ļoti patika/uzrunāja

5. dzejolis:

Vairāk par cilvēkiem pašiem
Es cilvēci mīlējis esmu,
Viņai man likās aizvien
Lielāks un vienīgais svars.
Cilvēki maksāja man
Ar to pašu - tie gāja man garām,
Kodols tiem likās mans darbs,
Atmesta čaumala - es.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

	1	2	3	4	5	6	7	8	9	10	
Nemaz nepatika/neuzrunāja	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Ļoti patika/uzrunāja

6. dzejolis:

Zvaigzne spīd debešos,
Mēness smaida joprojām.
Tikko bija tas riets,
Tagad nakts plaši staigā.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

	1	2	3	4	5	6	7	8	9	10	
Nemaz nepatika/neuzrunāja	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Ļoti patika/uzrunāja

7. dzejolis:

Mūžam dziņa un mūžam šaubas.
Mūžam meklēt un nerast nekad.
Gars un galva, un miesas, un sirds
Paši sevi saēd un sakvēlojas,
Un pelnus iznēsā vējš.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

1 2 3 4 5 6 7 8 9 10
Nemaz nepatika/neuzrunāja ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Ļoti patika/uzrunāja

8. dzejolis:

Laiks pēdas dzēš - bet pirmveids palicis,
Un dzīves koks gan atkal uzplaucis,
Bet jaunās lapas saista pērnais zibens.
Pat jūru nes uz pleciem drūmais dibens,
Kam pāri viļņo smaidošs līmenis,
Un laipnās gaismas bargais tēvs bij - zibens.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

1 2 3 4 5 6 7 8 9 10
Nemaz nepatika/neuzrunāja ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Ļoti patika/uzrunāja

9. dzejolis:

Saule spīd un visus silda,
Mīla, brīve visam cauri spīd.
Vieglas, skaidras visas sirdis,
Lielas, jautras visas domas.
Laikam, telpai pāri ceļas,
Sniegt un cerēt jaunu zemi.
Visiem līdzīgiem būt un labiem
Lai top katrs labs par sevi.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

1 2 3 4 5 6 7 8 9 10
Nemaz nepatika/neuzrunāja ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Ļoti patika/uzrunāja

10. dzejolis:

Mežs un putni
Un visa tāle
Mākoņu zaros
Pāri vārtiem
Ar jaunu valsti
Prom no bailēm
Vēl top, vēl skumst
Vēl jūt vienas sāpes

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

1 2 3 4 5 6 7 8 9 10
Nemaz nepatika/neuzrunāja ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Ļoti patika/uzrunāja

Veiktās aptaujas vidējie rezultāti

Aptauja "Datora ģenerētas dzejas novērtēšana"

Respondentu statistika:

Respondentu skaits	112
Vīrietis	25
Sieviete	87
Vidējais vecums	38.5 gadi

Rezultātu kopsavilkums:

1. dzejolis:

Smaids, kad vakars zied,
Es dziedu par Tevi,
Jo nav ko raudāt,
Nav ko skumt.
Nakts jau aiz loga nāk,
Laime kā lietus līst.
Es tūlīt jau nāku,
Tu mani dari savādāku.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja — Ļoti patika/uzrunāja

6.94 / 10

2. dzejolis:

Smiekli bez gala
Ritē un trīsē,
Līdz agram rītam
Līksmība laistās.
Dzīvības prieka
Pārpilnas sirdis
Pārplūst no katra
Niecīga vārdiņa.
Atbild uz katru
Smiekliem bez gala,
Izsauc no katra
Smieklus bez gala.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja — Ļoti patika/uzrunāja

6.35 / 10

3. dzejolis:

Tu smejies, met ar roku,
Tad klusi mājā ej uz dusu.
Lūk, ko tu neproti, draugs,
Tā jau zini, ka tā pati dvēsle plauks.
Es viņu leju katru brīdi
Ak, lai tik skaisti mirdz.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja — Ļoti patika/uzrunāja

5.94 / 10

4. dzejolis:

Vētra nāk un veļas,
Lietus arī ceļas.
Zūd Visas domas, krusa veļas,
Zūd saikne ar ikdienību un dūmi ceļas.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja  Ļoti patika/uzrunāja

6.75 / 10

5. dzejolis:

Vairāk par cilvēkiem pašiem
Es cilvēci mīlējis esmu,
Viņai man likās aizvien
Lielāks un vienīgais svars.
Cilvēki maksāja man
Ar to pašu - tie gāja man garām,
Kodols tiem likās mans darbs,
Atmesta čaumala - es.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja  Ļoti patika/uzrunāja

6.74 / 10

6. dzejolis:

Zvaigzne spīd debešos,
Mēness smaida joprojām.
Tikko bija tas riets,
Tagad nakts plaši staigā.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja  Ļoti patika/uzrunāja

6.12 / 10

7. dzejolis:

Mūžam dziņa un mūžam šaubas.
Mūžam meklēt un nerast nekad.
Gars un galva, un miesas, un sirds
Paši sevi saēd un sakvēlojas,
Un pelnus iznēsā vējš.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja  Ļoti patika/uzrunāja

6.4 / 10

8. dzejolis:

Laiks pēdas dzēš - bet pirmveids palicis,
Un dzīves koks gan atkal uzplaucis,
Bet jaunās lapas saista pērnais zibens.
Pat jūru nes uz pleciem drūmais dibens,
Kam pāri viļņo smaidošs līmenis,
Un laipnās gaismas bargais tēvs bij - zibens.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja  Ļoti patika/uzrunāja

6.2 / 10

9. dzejolis:

Saule spīd un visus silda,
Mīla, brīve visam cauri spīd.
Vieglas, skaidras visas sirdis,
Lielas, jautras visas domas.
Laikam, telpai pāri ceļas,
Sniegt un cerēt jaunu zemi.
Visiem līdzīgiem būt un labiem
Lai top katrs labs par sevi.

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja  Ļoti patika/uzrunāja

7.07 / 10

10. dzejolis:

Mežs un putni
Un visa tāle
Mākoņu zaros
Pāri vārtiem
Ar jaunu valsti
Prom no bailēm
Vēl top, vēl skumst
Vēl jūt vienas sāpes

Lūdzu novērtējiet dzejoli skalā no 1 līdz 10!

Nemaz nepatika/neuzrunāja  Ļoti patika/uzrunāja

6.63 / 10

Aptaujā iegūtie respondentu vērtējumi

3. pielikuma tabula

Dzejoļa nr. / Respondents	1	2	3	4	5	6	7	8	9	10
1	10	8	10	7	10	8	7	8	9	10
2	8	6	7	8	5	6	7	10	9	10
3	6	7	6	8	7	7	7	8	9	8
4	6	8	5	7	8	6	5	4	5	6
5	7	7	3	6	8	8	7	4	9	6
6	6	8	8	9	7	8	8	7	10	9
7	7	7	6	6	6	6	7	8	7	7
8	6	5	5	7	5	5	6	6	6	7
9	9	8	7	5	9	8	9	8	9	7
10	10	10	6	10	10	6	4	10	9	8
11	8	9	7	7	8	8	7	7	7	7
12	5	7	8	6	6	4	5	5	7	7
13	8	7	8	9	6	8	5	5	8	9
14	9	9	10	10	5	7	8	6	10	7
15	7	9	5	5	5	7	4	7	9	8
16	8	9	10	8	10	8	9	10	10	9
17	5	4	6	8	4	5	6	3	4	7
18	7	6	6	7	5	4	4	5	10	6
19	9	10	9	8	9	7	9	10	10	10
20	5	8	8	7	6	4	6	9	7	5
21	8	8	7	9	7	9	10	9	9	9
22	7	8	1	6	5	10	4	1	10	7
23	8	8	8	10	7	6	8	9	6	7
24	10	8	7	7	6	8	9	10	10	9
25	6	5	7	7	5	6	5	5	6	7
26	7	6	5	8	9	5	6	4	7	8
27	7	5	5	3	5	6	7	8	7	6
28	10	9	9	9	10	9	10	9	9	10
29	8	7	7	6	9	6	10	8	8	8
30	7	6	7	7	7	8	8	7	8	6
31	10	10	9	10	10	9	9	10	9	9
32	7	3	6	3	5	5	5	3	6	6
33	9	9	8	8	8	8	7	9	9	9

3. pielikuma tabulas turpinājums

Dzejoļa nr. / Respondents	1	2	3	4	5	6	7	8	9	10
34	10	8	5	4	3	7	5	6	7	3
35	8	7	8	8	9	8	8	7	9	10
36	10	5	4	4	5	3	5	6	4	6
37	8	6	6	9	5	5	7	4	8	5
38	5	5	7	7	6	6	8	8	9	9
39	4	6	3	5	7	5	4	5	9	4
40	5	4	4	4	4	5	5	6	6	6
41	6	3	3	4	8	5	3	2	4	1
42	10	9	9	8	9	8	10	9	9	10
43	5	4	4	5	5	4	5	6	5	6
44	5	3	2	3	7	6	9	6	5	1
45	9	8	6	7	6	8	5	7	9	8
46	7	8	5	6	7	6	7	8	8	7
47	6	9	5	8	10	2	3	7	10	4
48	3	5	5	8	3	6	2	2	6	6
49	8	9	7	7	7	8	7	8	10	8
50	9	6	6	7	9	8	8	6	3	3
51	8	4	3	9	6	5	3	8	10	6
52	10	10	9	9	8	8	6	7	7	7
53	6	4	5	6	5	6	7	6	4	6
54	5	5	5	5	6	6	5	5	5	5
55	3	5	4	6	4	6	3	5	5	6
56	1	5	6	6	8	4	3	6	7	5
57	7	9	8	9	10	7	10	10	10	10
58	7	4	4	9	2	4	5	4	7	8
59	8	8	8	10	9	10	8	8	10	9
60	5	6	1	6	6	6	1	6	8	6
61	6	7	8	6	6	6	4	3	8	6
62	4	6	5	4	6	6	5	4	5	6
63	10	10	10	10	10	10	10	10	10	10
64	6	7	6	7	8	7	7	7	8	6
65	7	4	5	8	5	6	7	5	7	6
66	7	6	4	1	8	1	8	2	2	4
67	8	7	4	4	8	6	5	4	8	8
68	8	8	9	9	6	6	4	7	9	6
69	5	8	3	5	5	7	6	7	8	4
70	8	9	7	10	9	6	8	8	8	9

3. pielikuma tabulas turpinājums

Dzejoļa nr. / Respondents	1	2	3	4	5	6	7	8	9	10
71	6	8	5	6	5	7	6	7	6	6
72	7	8	8	10	9	7	8	9	8	10
73	5	3	3	2	4	6	4	6	4	4
74	7	4	5	4	7	5	6	7	8	9
75	4	4	4	5	6	4	4	6	4	4
76	7	1	4	5	1	5	3	1	7	8
77	9	4	7	8	7	2	9	2	9	9
78	9	6	7	4	3	8	10	5	6	7
79	8	6	8	8	9	3	7	7	7	8
80	5	6	6	7	8	6	8	8	7	7
81	8	10	10	9	8	10	10	10	9	9
82	7	4	6	8	9	5	8	6	6	6
83	7	5	3	7	2	7	7	2	5	3
84	8	5	5	4	5	4	7	4	6	6
85	9	9	9	9	9	9	6	6	7	7
86	2	5	6	7	6	2	4	2	2	2
87	3	2	4	1	5	1	8	5	1	1
88	8	10	9	10	10	6	8	10	8	10
89	7	6	6	5	8	7	8	6	6	7
90	9	9	6	8	10	7	9	9	7	9
91	7	5	6	7	6	7	6	6	7	6
92	6	3	2	7	7	3	6	7	5	4
93	5	3	4	8	5	2	7	4	7	2
94	3	7	8	2	8	1	5	2	2	2
95	7	4	3	9	8	4	3	4	4	7
96	8	4	4	7	6	4	3	2	3	3
97	7	2	3	4	5	7	7	5	3	3
98	8	2	3	6	2	6	4	7	6	5
99	8	5	6	7	6	5	4	4	6	6
100	3	4	4	6	6	5	5	5	5	6
101	7	6	8	6	7	9	5	7	9	4
102	6	3	7	4	9	3	5	5	6	5
103	7	6	7	6	6	8	8	7	7	6
104	8	5	7	8	6	6	6	7	9	8
105	7	8	5	6	7	8	9	5	8	8
106	5	8	5	9	6	7	7	4	7	4
107	4	4	4	9	8	5	6	6	5	7

3. pielikuma tabulas turpinājums

Dzejoļa nr. / Respondents	1	2	3	4	5	6	7	8	9	10
108	6	3	3	5	8	5	9	3	4	6
109	10	9	9	9	7	9	9	8	10	10
110	9	10	8	9	9	8	10	10	9	10
111	9	7	6	6	8	6	7	7	8	9
112	5	7	3	5	7	8	2	4	4	5

Ticamības intervālu, standartnovirzes un standartkļūdas aprēķini

4. pielikuma tabula

Dzejoļa numurs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Vidējais rezultāts	6.938	6.348	5.938	6.750	6.741	6.116	6.402	6.196	7.071	6.625
Standartnovirze	1.946	2.225	2.124	2.103	2.030	2.008	2.141	2.324	2.172	2.255
Standartkļūda	0.184	0.210	0.201	0.199	0.192	0.190	0.202	0.220	0.205	0.213
95% tic. int.	0.360	0.412	0.393	0.390	0.376	0.372	0.397	0.430	0.402	0.418

Maģistra darbs “Datora ģenerēta dzeja” izstrādāts LU Datorikas fakultātē.

Darba teksta galīgā versija izgatavota 23.05.2022**.**

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Laura Jozefa, 23.05.2022
(Autora paraksts un datums)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par **p i e m ē r o t u / n e p i e m ē r o t u** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Signature not validated

Darba vadītājs: _____
(Vadītāja paraksts un datums)

Digitally signed by KASPARS BALODIS
Date: 2022.05.23 13:48:22 EEST

Darbs iesniegts **maģistratūras sekretariātā** _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____.
(Metodiķes paraksts)

Recenzents: Asociētais profesors, Dr. sc. comp. Normunds Grūzītis
(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____prot. Nr. _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)