ხედვის კოგნიტური აღწერის მოდელი ქართულ ენაზე

ავტორი: დავით ბიტმალკიშევ

სადისერტაციო ნაშრომი შესრულებულია საქართველოს უნივერსიტეტის მეცნიერების და ტექნოლოგიების სკოლაში კომპიუტერული მეცნიერების დოქტორის ხარისხის მოსაპოვებლად

ხელმძღვანელი: პროფესორი სერგო ცირამუა

თბილისი

2026

მიძღვნა

ეძღვნება დედაჩემს.

მიმოხილვა

თანამედროვე მსოფლიოს ერთ-ერთ ყველაზე მნიშვნელოვან გამოწვევას წარმოადგენს ხელოვნური ინტელექტის მოდელების მრავალენოვანი და კულტურულად ადაპტირებული გამოყენება. მიუხედავად იმისა, რომ გლობალურად მრავალი წარმატებული ვიზუალური აღწერის სისტემა არსებობს (მაგალითად, BLIP2, Flamingo, GPT-4V), ისინი ძირითადად ინგლისურენოვანი მონაცემების საფუძველზეა გაწვრთნილი. ამან გამოიწვია სერიოზული გარღვევა ტექნოლოგიაში, თუმცა, მცირე ენობრივი საზოგადოებები, როგორიცაა ქართული, კვლავაც დაუცველია ხელოვნური ინტელექტის მომსახურებების მიმართ.

ამ კონტექსტში განსაკუთრებულ მნიშვნელობას იძენს „Martha“ — კომპიუტერული ხედვისა და ბუნებრივი ენის გენერაციის ჰიბრიდული მოდელი, რომელიც სპეციალურად ქართულ ენაზე ახორციელებს ვიზუალური გარემოს აღწერას. პროექტი აერთიანებს ორ უმნიშვნელოვანეს კომპონენტს: BLIP2-ის ვიზუალურ ტრანსფორმერს, რომელიც სურათებიდან მაღალი დონის სემანტიკურ ტენსორებს აგენერირებს, და ByT5-ის ენობრივ ტრანსფორმერს, რომელიც მონაცემებს ქართულ ენაზე აღწერს. სწორედ ამ სინთეზის მეშვეობით იქმნება სრულიად ახალი შესაძლებლობა: გარემოს ავტომატური აღქმა და აღწერა ქართულ ენაზე.

„Martha“ არ არის მხოლოდ ტექნოლოგიური ინოვაცია — ის არის სოციალურად და კულტურულად მნიშვნელოვანი პროექტი, რომელიც პასუხობს გლობალურ გამოწვევას: როგორ გავხადოთ ხელოვნური ინტელექტი თანასწორი, მრავალენოვანი და ყველასთვის ხელმისაწვდომი. მისი მნიშვნელობა საქართველოსთვის უდიდესია, რადგან ქმნის ახალი თაობის ქართულენოვან ტექნოლოგიას, ხოლო სიახლე მდგომარეობს იმაში, რომ პირველად განხორციელდა მაღალტექნოლოგიური მოდელების ჰიბრიდიზაცია ქართულ ენაზე ვიზუალური გარემოს აღწერისთვის.

სარჩევი

გამოყენებული ცხრილების და ილუსტრაციების ჩამონათვალი

ტერმინების განმარტება

AdamW

ოპტიმიზაციის ალგორითმი, რომელიც აერთიანებს ადაპტიურ სწავლების სიჩქარეს და რეგულარიზაციის მექანიზმს (Weight Decay), რაც ხელს უწყობს სტაბილურ და ეფექტურ სწავლებას.

Artificial Intelligence (ხელოვნური ინტელექტი)

კომპიუტერული მეცნიერების დარგი, რომელიც შეისწავლის ისეთი სისტემების შექმნას, რომლებიც ადამიანის მსგავსად აღიქვამენ, სწავლობენ, აანალიზებენ და იღებენ გადაწყვეტილებებს.

Attention (ყურადღების მექანიზმი)

მექანიზმი ნეირონულ ქსელში, რომელიც განსაზღვრავს, თუ რომელი ინფორმაციაა ყველაზე მნიშვნელოვანი მოცემულ კონტექსტში.

Augmentation (მონაცემთა გაფართოება)

სასწავლო მონაცემების ხელოვნურად გამრავალფეროვნება (მაგალითად, სურათის გადატრიალება, ზომის შეცვლა), რათა გაიზარდოს მოდელის გამძლეობა.

Backpropagation (უკუსვლის ალგორითმი)

სწავლის პროცესი, რომლის დროსაც დანაკარგის ფუნქციის საფუძველზე იცვლება ნეირონული ქსელის წონები.

Batch Size

სასწავლო პროცესში ერთდროულად დამუშავებული მონაცემების რაოდენობა.

Bias (წაძვრა)

დამატებითი პარამეტრი ნეირონში, რომელიც ემატება შეწონილ ჯამს და ზრდის მოდელის მოქნილობას.

BLEU

ავტომატური შეფასების მეტრიკა, რომელიც განსაზღვრავს გენერირებული ტექსტის შესაბამისობას ეტალონურ ტექსტთან.

BLIP2

ვიზუალურ-ენობრივი მოდელი (Bootstrapping Language-Image Pretraining), რომელიც სურათიდან იღებს სემანტიკურ წარმოდგენას და ამზადებს მას ტექსტური გენერაციისთვის.

Byte Pair Encoding (BPE)

ტოკენიზაციის ალგორითმი, რომელიც ტექსტს შლის ქვესიტყვოვან ერთეულებად.

ByT5

ენობრივი ტრანსფორმერული მოდელი, რომელიც ტექსტს ბაიტურ დონეზე ამუშავებს და არ საჭიროებს ტოკენიზაციას.

CIDEr

მეტრიკა, რომელიც გამოიყენება სურათის აღწერის ხარისხის შესაფასებლად.

Cloud Computing (ღრუბლოვანი გამოთვლები)

ინტერნეტზე დაფუძნებული გამოთვლითი რესურსების გამოყენება (მაგალითად, Google Cloud, AWS).

CNN (Convolutional Neural Network)

კონვოლუციური ნეირონული ქსელი, რომელიც გამოიყენება სურათებისა და სივრცითი მონაცემების დასამუშავებლად.

Cross-Entropy

დანაკარგის ფუნქცია, რომელიც გამოიყენება კლასიფიკაციის ამოცანებში პროგნოზის სიზუსტის შესაფასებლად.

Dataset (მონაცემთა ბაზა)

სტრუქტურირებული მონაცემების ერთობლიობა, რომელიც გამოიყენება მოდელის სწავლებისა და შეფასებისთვის.

Decoder

Encoder–Decoder არქიტექტურის ნაწილი, რომელიც შიდა წარმოდგენას გარდაქმნის საბოლოო ტექსტურ გამოსავლად.

Dropout

რეგულარიზაციის მეთოდი, რომლის დროსაც სწავლის პროცესში დროებით ითიშება ნეირონების ნაწილი.

Encoder

ნეირონული არქიტექტურის ნაწილი, რომელიც შეყვანილ მონაცემს გარდაქმნის სემანტიკურ წარმოდგენად.

Encoder–Decoder არქიტექტურა

სტრუქტურა, სადაც ერთი მოდული ქმნის მონაცემის შიდა წარმოდგენას (Encoder), ხოლო მეორე მას გარდაქმნის გამოსავლად (Decoder).

Epoch

სასწავლო ციკლი, რომლის განმავლობაში მოდელი ერთხელ ამუშავებს მთელ სასწავლო მონაცემთა ბაზას.

Exploding Gradient

პრობლემა, როდესაც გრადიენტები სწავლის პროცესში ზედმეტად იზრდება და სწავლა არასტაბილური ხდება.

Feature Map (მახასიათებლების რუკა)

კონვოლუციური ოპერაციის შედეგად მიღებული მატრიცა, რომელიც ასახავს ამოცნობილ მახასიათებლებს.

Fine-Tuning

უკვე გაწვრთნილი მოდელის დამატებითი გადამზადება კონკრეტულ მონაცემებზე.

Flattening (გაბრტყელება)

მრავალგანზომილებიანი მონაცემის ერთგანზომილებიან ვექტორად გარდაქმნა.

Fully Connected Layer

ფენა, სადაც თითოეული ნეირონი დაკავშირებულია წინა ფენის ყველა ნეირონთან.

GPU

გრაფიკული პროცესორი, რომელიც გამოიყენება მასშტაბური პარალელური გამოთვლებისთვის.

Gradient Clipping

მეთოდი, რომელიც ზღუდავს გრადიენტის მაქსიმალურ მნიშვნელობას სწავლის სტაბილურობის უზრუნველსაყოფად.

Gradient Descent (გრადიენტული დაშვება)

ოპტიმიზაციის მეთოდი, რომელიც დანაკარგის ფუნქციის მინიმუმს ეძებს.

Hyperparameter (ჰიპერპარამეტრი)

მოდელის პარამეტრი, რომელიც წინასწარ განისაზღვრება (მაგალითად, სწავლების სიჩქარე).

Image Preprocessing

სურათის წინასწარი დამუშავება (ნორმალიზაცია, მასშტაბირება და სხვა).

Learning Rate (სწავლების სიჩქარე)

პარამეტრი, რომელიც განსაზღვრავს, რამდენად სწრაფად იცვლება წონები სწავლის პროცესში.

LSTM (Long Short-Term Memory)

რეკურენტული ნეირონული ქსელის გაუმჯობესებული ვერსია, რომელიც უკეთ ინახავს გრძელვადიან დამოკიდებულებებს.

Loss Function (დანაკარგის ფუნქცია)

მათემატიკური ფუნქცია, რომელიც ზომავს პროგნოზსა და რეალურ მნიშვნელობას შორის სხვაობას.

LoRA

მოდელის პარამეტრების შემცირების მეთოდი, რომელიც ამცირებს გამოთვლით რესურსებს.

METEOR

ტექსტის ხარისხის შეფასების მეტრიკა, რომელიც ითვალისწინებს სინონიმებსა და მნიშვნელობრივ მსგავსებას.

Multimodal (მულტიმოდალური სისტემა)

სისტემა, რომელიც ერთდროულად ამუშავებს სხვადასხვა ტიპის მონაცემებს (მაგალითად, სურათი და ტექსტი).

NLP (Natural Language Processing)

ბუნებრივი ენის დამუშავება — კომპიუტერული მეთოდები ადამიანის ენის ანალიზისა და გენერაციისთვის.

Optimizer (ოპტიმიზატორი)

ალგორითმი, რომელიც არეგულირებს წონების განახლებას სწავლის პროცესში.

Overfitting (გადატრენინგება)

მდგომარეობა, როდესაც მოდელი ზედმეტად ერგება სასწავლო მონაცემებს და ვერ მუშაობს ახალ მონაცემებზე.

Padding

სურათის კიდეებზე დამატებული ნულოვანი მნიშვნელობები, რათა ზომა არ შემცირდეს კონვოლუციის დროს.

Patch (პატჩი)

სურათის მცირე ბლოკი, რომელსაც ტრანსფორმერი ცალკე ამუშავებს.

Pooling

ოპერაცია, რომელიც ამცირებს მახასიათებლების რუკის ზომას.

Projection Layer (პროექციის შრე)

ფენა, რომელიც გარდაქმნის ერთ ვექტორულ სივრცეს მეორეში.

Q-Former (Query Transformer)

BLIP2-ის კომპონენტი, რომელიც ვიზუალურ ინფორმაციას აკომპაქტებს ენობრივი მოდელისთვის.

Quantization

მოდელის ზომის შემცირების მეთოდი, რომელიც ამცირებს მეხსიერების მოთხოვნას.

RNN (Recurrent Neural Network)

რეკურენტული ნეირონული ქსელი, რომელიც მუშაობს მიმდევრობით მონაცემებზე.

Self-Attention (თვითყურადღება)

მექანიზმი, რომელიც განსაზღვრავს მიმდევრობის ელემენტებს შორის ურთიერთმნიშვნელობას.

Sigmoid

აქტივაციის ფუნქცია, რომლის შედეგი 0-დან 1-მდე დიაპაზონშია.

Softmax

ფუნქცია, რომელიც გარდაქმნის რიცხვებს ალბათობებად.

StandardScaler

მონაცემთა ნორმალიზაციის მეთოდი, რომელიც ცენტრსა და დისპერსიას ასწორებს.

Stride

ფილტრის გადაადგილების ბიჯი კონვოლუციის დროს.

Tokenization (ტოკენიზაცია)

ტექსტის დაშლა მცირე ერთეულებად (სიტყვებად ან ქვესიტყვებად).

Transformer

ნეირონული არქიტექტურა, რომელიც ეფუძნება ყურადღების მექანიზმს.

TTS (Text-to-Speech)

ტექსტის ხმოვან ფორმატში გარდაქმნის ტექნოლოგია.

UTF-8

სიმბოლოების უნივერსალური კოდირების სტანდარტი.

Validation Set

მონაცემთა ნაწილი, რომელიც გამოიყენება მოდელის შუალედური შეფასებისთვის.

Vanishing Gradient

პრობლემა, როდესაც გრადიენტი ძალიან მცირე ხდება და სწავლა ფერხდება.

Vision Transformer (ViT)

ტრანსფორმერული არქიტექტურა, რომელიც გამოიყენება სურათების დასამუშავებლად.

შესავალი

თემის აქტუალობა

კვლევის აქტუალობა რამდენიმე ძირითად გამოწვევას უკავშირდება:

1. გლობალური უთანასწორობა ენებს შორის – მსოფლიო AI ინდუსტრიაში ინგლისურენოვანი რესურსების დომინაცია მცირე ენებს ჩრდილავს. ამდენად, ქართული ენის ტექნოლოგიური ინტეგრაცია პირდაპირ პასუხობს ციფრული სუვერენიტეტისა და ენობრივი მრავალფეროვნების გამოწვევას.
2. ინკლუზიურობის დეფიციტი – საერთაშორისო მოდელების უმეტესობა არ არის ადაპტირებული ქართულ ენაზე მუშაობისთვის, რაც მნიშვნელოვნად ზღუდავს მხედველობადაქვეითებული და არასახელმწიფო ენების მომხმარებლების უფლებებს.
3. ტექნოლოგიური უსაფრთხოება – ადგილობრივ ენებზე მომუშავე სისტემების არსებობა ქვეყნისთვის სტრატეგიული მნიშვნელობისაა, რადგან უზრუნველყოფს მონაცემების ლოკალურ დამუშავებასა და უსაფრთხოებას.
4. ეკონომიკური პოტენციალი – ქართულ ენაზე მომუშავე AI ტექნოლოგიების განვითარება პირდაპირ უწყობს ხელს სტარტაპ ეკოსისტემის გაძლიერებას და საინფორმაციო ტექნოლოგიების გლობალურ ბაზარზე საქართველოს ჩართვას.

პროექტის ღირებულება

„Martha“-ს ღირებულება განისაზღვრება რამდენიმე თვალსაზრისით:

1. ენობრივი დამოუკიდებლობა – პროექტი ქმნის ტექნოლოგიას, რომელიც ქართულ ენაზე მომუშავე მომხმარებლებს საშუალებას აძლევს, მიიღონ ზუსტი, ბუნებრივი და კონტექსტუალურად ადეკვატური აღწერები.
2. ინკლუზიური ტექნოლოგია – ერთ-ერთი ყველაზე მნიშვნელოვანი ასპექტი არის შშმ პირთა, კერძოდ, მხედველობადაქვეითებულთა დახმარება. სისტემა გარემოს აღწერით აუდიო არხზე გადაცემას უზრუნველყოფს, რითაც ზრდის მათ დამოუკიდებელ ცხოვრებასა და საზოგადოებრივ ინტეგრაციას.
3. საგანმანათლებლო და კულტურული მნიშვნელობა – ქართულ ენაზე სურათების აღწერის შესაძლებლობა მნიშვნელოვნად წაახალისებს როგორც ქართული მონაცემთა ბაზების შექმნას, ისე ადგილობრივ საგანმანათლებლო და სამეცნიერო რესურსებს.
4. ბიზნეს-ინოვაციური ღირებულება – სისტემა შეიძლება დაინერგოს სხვადასხვა სექტორში: უსაფრთხოებაში (ვიდეო-მონიტორინგი ქართულ ენაზე), მედიცინაში

(ვიზუალური დიაგნოსტიკა და დოკუმენტირება), ტურიზმში (ავტომატური გიდის ტექნოლოგია), მედიასა და განათლებაში.

სამეცნიერო კვლევის თეორიული სიახლე

შემოთავაზებული მეთოდოლოგია, რომელიც აერთიანებს ვიზუალურ ტრანსფორმერებსა და ბაიტურ დონეზე მოქმედ ენობრივ მოდელს (ByT5), ქმნის საიმედო საფუძველს ქართულენოვანი კომპიუტერული ხედვის სისტემების განვითარებისათვის. ნაშრომი მნიშვნელოვანია როგორც თეორიული, ისე პრაქტიკული თვალსაზრისით, რადგან იგი ხელს უწყობს თანამედროვე და ინკლუზიური ციფრული ეკოსისტემის ჩამოყალიბებას.

კვლევის მთავარი სამეცნიერო სიახლე მდგომარეობს BLIP2-ის ვიზუალური კომპონენტის და ByT5 ენობრივი მოდელის ჰიბრიდულ ინტეგრაციაში, რომელიც მიმართულია ქართულენოვანი ტექსტის გენერაციის ამოცანის გადაწყვეტისკენ. კერძოდ:

- ქართულ ენაზე ვიზუალური სცენის აღწერისთვის პირველადაა წარმოდგენილი encoder–decoder არქიტექტურაზე დაფუძნებული ერთიანი სისტემა;
- შემუშავებულია ვიზუალური ემბედინგების პროექციის მექანიზმი, რომელიც უზრუნველყოფს მათ სინქრონიზაციას ByT5 მოდელის დამალულ წარმოდგენებთან;
- შექმნილია სპეციალიზებული ქართულენოვანი ვიზუალურ-ტექსტური (Image Captioning) მონაცემთა ბაზა, რომელიც აერთიანებს გამოსახულებებსა და შესაბამის აღწერებს;
- დასაბუთებულია ბაიტურ დონეზე მომუშავე მოდელის უპირატესობა მორფოლოგიურად მდიდარი ენის შემთხვევაში.

კვლევის მეთოდოლოგია ეფუძნება ექსპერიმენტულ და შედარებით-ანალიტიკურ მიდგომებს, რაც სრულად შეესაბამება დასახულ მიზნებს. კვლევის პროცესში განხორციელდა მონაცემთა ბაზის ფორმირება, მონაცემების წინასწარი დამუშავება, მოდელის არქიტექტურული დაგეგმვა, მისი ტრენინგი და მიღებული შედეგების შეფასება. გამოყენებულია როგორც რაოდენობრივი შეფასების ინსტრუმენტები, ისე სხვადასხვა სცენარში ჩატარებული ექსპერიმენტული ტესტირება. მეთოდოლოგიური ჩარჩო თანმიმდევრულად და ლოგიკურად ასახავს კვლევის ეტაპებს. ლიტერატურის მიმოხილვა მოიცავს თანამედროვე ვიზუალურ-ენობრივი მოდელების განხილვას და ქმნის მყარ თეორიულ საფუძველს წარმოდგენილი არქიტექტურისთვის, თუმცა სასურველი იქნებოდა უახლესი საერთაშორისო კვლევების უფრო ფართო და სიღრმისეული შედარებითი ანალიზი.

სამეცნიერო კვლევის პრაქტიკული სიახლე

„Martha“-ს კვლევითი სიახლე რამდენიმე განზომილებით გამოიხატება:

1. მოდელების ინოვაციური სინთეზი – პროექტი პირველად ახორციელებს BLIP2-ის ვიზუალური ტენსორების ByT5-ის ენობრივ არხთან სინქრონიზაციას სპეციალურად ქართული ენისთვის.
2. ბაიტ-დონის დამუშავება – ByT5-ის უნიკალური არქიტექტურა (რომელსაც არ აქვს ტოკენიზატორი) საშუალებას იძლევა, ქართულ ენაზე მაღალი ხარისხის ტექსტი გენერირდეს, რაც სხვა მოდელებისთვის პრობლემურია.
3. ქართული მონაცემთა ბაზის შექმნა – პროექტის ფარგლებში მუშავდება სპეციალური ქართული სურათ-კაპშენის მონაცემთა ბაზა, რაც პირველი შემთხვევაა ამ მასშტაბით.
4. მრავალმხრივი გამოყენებადობა – ტექნოლოგია არა მხოლოდ სამეცნიერო სფეროს ემსახურება, არამედ სარგებლიანია საზოგადოებრივ, კომერციულ და სახელმწიფო სერვისებში.
5. სოციალური ინოვაცია – ინკლუზიური ტექნოლოგიის დანერგვა ქართულ ენაზე ხელს უწყობს თანასწორობის ზრდას და ქმნის რეალურ შესაძლებლობებს შშმ პირებისთვის.

შედეგების მნიშვნელობა

კვლევის შედეგებს ექნება როგორც ეროვნული, ასევე რეგიონული და გლობალური გავლენა:

1. საქართველოსთვის – პროექტი ხელს შეუწყობს ქართულ ენაზე მომუშავე ხელოვნური ინტელექტის ეკოსისტემის განვითარებას, სტარტაპებისა და უნივერსიტეტების თანამშრომლობას, მონაცემთა ლოკალური ბაზების შექმნას.
2. რეგიონისთვის – „Martha“ შეიძლება გახდეს მოდელი სამხრეთ კავკასიისა და მცირე ენების საზოგადოებისთვის, სადაც მსგავსი გამოწვევები არსებობს.
3. მსოფლიოსთვის – კვლევა ამტკიცებს, რომ ხელოვნური ინტელექტის მრავალენოვანი განვითარება შესაძლებელია მცირე ენებისთვისაც, რაც ენების თანასწორობისა და კულტურული მრავალფეროვნების შენარჩუნებას ემსახურება.

კვლევის მიზანი

პროექტის მთავარი მიზანია ისეთი კომპიუტერული ხედვისა და ენობრივი გენერაციის ჰიბრიდული სისტემის შექმნა, რომელიც ქართულ ენაზე მოახდენს ვიზუალური გარემოს ზუსტ და ბუნებრივ აღწერას. „Martha“-ს მიზანია ქართული ენის ხელოვნური ინტელექტის ეკოსისტემაში ინტეგრაცია, ინკლუზიური ტექნოლოგიების ხელშეწყობა და ქვეყნისთვის სტრატეგიულად მნიშვნელოვანი ენობრივი დამოუკიდებლობის განმტკიცება.

კვლევის ამოცანები და ეტაპები

1. მონაცემთა ბაზის შექმნა და მომზადება

   ამოცანა: ქართული სურათ-კაპშენის მონაცემთა ბაზის აგება (სურათებისა და შესაბამისი ტექსტური აღწერების ფორმირება).

   მოსალოდნელი შედეგი: მრავალფეროვანი, სტრუქტურირებული და სანდო მონაცემთა ნაკრები, რომელიც საფუძვლად დაედება მოდელის გაწვრთნას.

2. BLIP2 ვიზუალური მოდულის ადაპტაცია

   ამოცანა: BLIP2 მოდელის გამოყენება ვიზუალური ტენსორების გენერირებისთვის და მისი ინტეგრაცია ქართულენოვან არქიტექტურაში.

   მოსალოდნელი შედეგი: გარემოს ვიზუალური ობიექტების, კონტექსტისა და სემანტიკური ურთიერთობების მაღალი სიზუსტით ამოცნობა.

3. ByT5 ენობრივი მოდულის ოპტიმიზაცია

   ამოცანა: ByT5 მოდულის ინტეგრაცია ისე, რომ ვიზუალური ტენსორები ქართულ ტექსტად გარდაქმნას.

   მოსალოდნელი შედეგი: ბუნებრივი, გრამატიკულად და სტილისტურად სწორი ქართულენოვანი აღწერების გენერაცია.

4. მოდელების სინქრონიზაცია და ჰიბრიდული არქიტექტურის შექმნა

   ამოცანა: ვიზუალური და ენობრივი მოდულების დაკავშირება ერთიან სისტემად (encoder–decoder სტრუქტურა).

   მოსალოდნელი შედეგი: ქართული ენისთვის სპეციალურად მორგებული ჰიბრიდული მოდელი, რომელსაც შეუძლია გარემოს აღწერა.

5. ინკლუზიური ფუნქციონალის შემუშავება

   ამოცანა: მოდელის შედეგების აუდიო ფორმატში გარდაქმნა მხედველობადაქვეითებულთათვის.

მოსალოდნელი შედეგი: პრაქტიკული აპლიკაცია, რომელიც რეალურ სოციალურ გამოწვევებს მოაგვარებს.

6. ტესტირება, შეფასება და ვალიდაცია

   ამოცანა: მოდელის მუშაობის შეფასება სხვადასხვა სცენარებში (საგანმანათლებლო, სამედიცინო, ტურიზმი, უსაფრთხოება).

   მოსალოდნელი შედეგი: დადასტურებული სიზუსტე, რეალური გამოყენებადობა და პრაქტიკული ღირებულება.

მოსალოდნელი შედეგები

1. ქართული სურათ-კაპშენის მონაცემთა ბაზა.
2. BLIP2 + ByT5-ის ინტეგრაციაზე დაფუძნებული ჰიბრიდული მოდელი.
3. ქართული ენაზე გარემოს აღწერის უნარი.
4. ინკლუზიური გადაწყვეტა შშმ პირთათვის.
5. ქართული ენის ხელოვნური ინტელექტის ეკოსისტემის გაძლიერება და საერთაშორისო ცოდნის ბაზარზე მისი ინტეგრაცია.

კვლევის მეთოდოლოგია

ზოგადი მიდგომა

„Martha“-ს კვლევა ეფუძნება კომპიუტერული ხედვისა და ბუნებრივი ენის დამუშავების თანამედროვე ინტეგრირებულ მიდგომას. პროექტის დიზაინი ეფუძნება encoder–decoder არქიტექტურას, სადაც ვიზუალური ნაწილი (BLIP2) უზრუნველყოფს სემანტიკური ტენსორების გენერირებას სურათებიდან, ხოლო ენობრივი ნაწილი (ByT5) ამ ინფორმაციას ქართულ ენაზე გარდაქმნის.

მეთოდოლოგია მრავალეტაპიანია და მოიცავს: მონაცემთა შეგროვებასა და წინამუშავებას, მოდელების ადაპტაციასა და სინქრონიზაციას, ფუნქციური მოდულის შექმნას, ტესტირებასა და შედეგების ვალიდაციას.

კვლევის ეტაპები

1. მონაცემთა შეგროვება და წინამუშავება

   ქართული სურათ-კაპშენის მონაცემთა ბაზის შექმნა (საჯარო და ლოკალური მონაცემების ინტეგრაცია).

ტექსტების წინამუშავება, ორთოგრაფიული და სემანტიკური ხარისხის კონტროლი.

სურათების სტანდარტიზაცია (გადამუშავება, ზომის ოპტიმიზაცია).

2. მოდელების ადაპტაცია

   BLIP2 ვიზუალური encoder-ის გამოყენება, რომელიც უკვე გაწვრთნილია მრავალენოვან სურათ-ტექსტურ მონაცემებზე.

   ByT5 decoder-ის ოპტიმიზაცია ქართულ ენაზე, ბაიტ-დონის დამუშავების უპირატესობით.

3. ჰიბრიდული არქიტექტურის სინქრონიზაცია

   ვიზუალური და ენობრივი მოდულების დაკავშირება ერთიან encoder–decoder სისტემაში.

   ტენსორების სივრცის ოპტიმიზაცია ისე, რომ ByT5-ის გენერირებული ტექსტი იყოს ბუნებრივი და კონტექსტუალურად შესაბამისი.

4. ინკლუზიური ფუნქციონალის დანერგვა

   ტექსტის აუდიოში გარდაქმნა (TTS) მხედველობადაქვეითებული მომხმარებლებისთვის.

   ტესტირება რეალურ სცენარებში: საგანმანათლებლო, ტურისტული, სამედიცინო და ყოველდღიური გამოყენების კონტექსტში.

5. შეფასება და ვალიდაცია

   მომხმარებელთა ტესტირება (human evaluation) ქართულ ენაზე ტექსტების ბუნებრიობისა და სიზუსტის დასადასტურებლად.

მეთოდოლოგიის შესაბამისობა მიზნებთან

ეს მიდგომა პირდაპირ პასუხობს პროექტის მთავარ მიზანს — შექმნას ჰიბრიდული მოდელი, რომელიც ქართულ ენაზე აღწერს ვიზუალურ გარემოს. მრავალეტაპიანი დიზაინი უზრუნველყოფს როგორც მეცნიერულ სიზუსტეს (საბაზისო მოდელების სინთეზი და მეტრიკებით შეფასება), ისე პრაქტიკულ შედეგს (ინკლუზიური აპლიკაცია რეალური მომხმარებლისთვის).

შეზღუდვები და უპირატესობები

შეზღუდვები:

1. ქართული მონაცემების სიმცირე (შედარებით მცირე კორპუსი ინგლისურთან).

2. მაღალი გამოთვლითი რესურსების საჭიროება (GPU, დიდი მეხსიერება).
3. ქართულ ტექსტში მრავალი მორფოლოგიური თავისებურება, რაც გენერაციის ხარისხს ართულებს.

უპირატესობები:

1. ByT5-ის უნიკალური არქიტექტურა, რომელიც ტოკენიზაციის გარეშე მუშაობს და ქართულ ენას ბუნებრივად ამუშავებს.
2. BLIP2-ის უნივერსალური encoder, რომელიც მდიდარ ვიზუალურ წარმოდგენებს ქმნის.
3. მრავალსაფეხურიანი ტესტირება უზრუნველყოფს შედეგების სანდოობასა და გამოყენებადობას.
4. ინკლუზიურობა და სოციალური გავლენა ტექნოლოგიის დანერგვისას.

მოსალოდნელი რისკები, წინაღობები და მათი შემცირების გზები

„Martha“-ს პროექტის განხორციელებისას მოსალოდნელია რიგი რისკები და პრობლემები, რომელთა გათვალისწინება და მართვა კრიტიკულად მნიშვნელოვანია.

1. მონაცემთა სიმცირე და ხარისხი – ქართულ ენაზე სურათ-კაპშენის კორპუსი შეზღუდულია.

   შემცირების გზა: არსებული საერთაშორისო მონაცემთა ბაზების ადაპტაცია, crowdsourcing-ით ქართული კაპშენების გენერაცია, ლინგვისტების ჩართვა ხარისხის კონტროლში.

2. მაღალი გამოთვლითი რესურსების საჭიროება – BLIP2 და ByT5 დიდ რესურსებს მოითხოვს, რაც პროცესს აძვირებს.

   შემცირების გზა: ღრუბლოვანი სერვისების (Google Cloud, AWS) გამოყენება, მოდელის პარამეტრების ოპტიმიზაცია და LoRA/quantization მეთოდების დანერგვა. (Ha, Shen, Wallis, Allen-Zhu, Li, Wang, Wang,Chen / ჰა, შენ, ვალისი,ალენ-ჟუ,ლი,ლი,ვანგ,ჩენ. 2021)

3. ენობრივი სპეციფიკა – ქართული ენის მორფოლოგიური სირთულეები შესაძლოა გენერაციის სიზუსტეს შეაფერხოს.

   შემცირების გზა: დამატებითი fine-tuning ქართულ მონაცემებზე, ადამიანის შეფასების (human evaluation) ინტეგრაცია. (Hodosh , Young,Hockenmaier/ჯოდოშ,იანგ,ჰოკერმაიერ. 2013)

4. ინკლუზიური ფუნქციონალის ტესტირების სირთულე – მხედველობადაქვეითებული პირების ჩართვა შესაძლოა ადმინისტრაციულ და ეთიკურ ბარიერებს წააწყდეს.

   შემცირების გზა: თანამშრომლობა ადგილობრივ NGO-ებთან და წინასწარი ეთიკური თანხმობის უზრუნველყოფა.

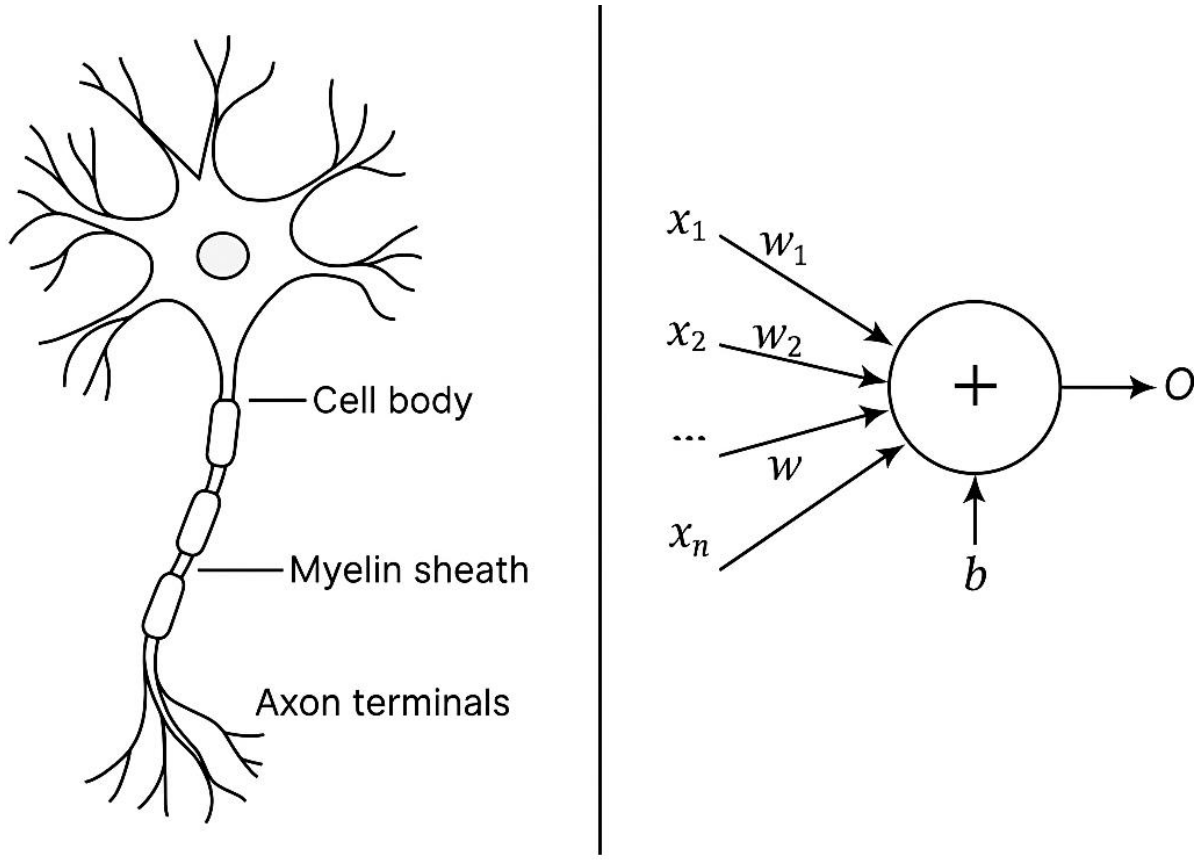
5. ფინანსური და დროში გადაცილების რისკი – მაღალი ტექნოლოგიური ღირებულება და რესურსების დაგვიანება.

   შემცირების გზა: ეტაპობრივი ბიუჯეტირება, ალტერნატიული დაფინანსების წყაროების მოძიება, პარალელური სამუშაოების დაგეგმვა.

თავი I

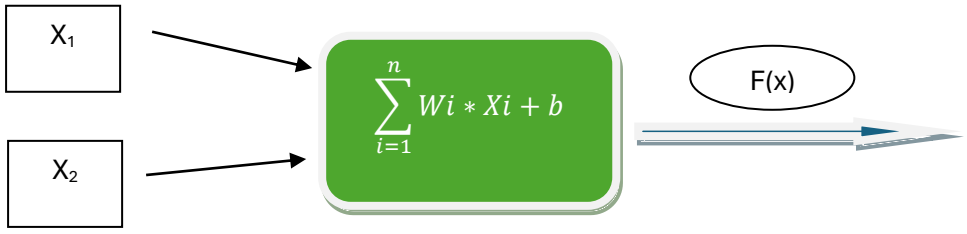
1.1 ნეირონული ქსელები

რა არის ნეირონი ? როგორ მუშაობს იგი ? ხშირ შემთხვევაში ხელოვნურ ნეირონს ადარებენ ბიოლოგიურს. მასაც ბიოლოგიურის მსგავსად აქვს დენდრიტები, ნუკლეუსი, აქსონები. დენდტრიტებიდან ნეირონი იღებს მონაცემს, ამუშავებს ნუკლეუსში და შემდგომ აქსონის ტერმინალებიდან გამოსცემს პასუხს, რომელსაც სხვა ნეირონს გადასცემს დენდტრიტების საშუალებით. ორ ან რამდენიმე ნეირონის კავშირს სინაპსს უწოდებენ.



გრაფიკი 1. ბიოლოგიური და ხელოვნური ნეირონები

ბიოლოგიურისგან განსხვავებით ხელოვნურ ნეირონს რათქმაუნდა არ გააჩნია “ცოცხალი” ფუნქციები, ამიტომ საჭიროა მათემატიკური მიდგომები, რათა მან შექმნას ბიოლოგიური ნეირონის იმიტაცია. ნეირონი არის ერთგვარი შავი ყუთი, რომელიც შემავალ მონაცემებზე ზემოქმედებს გარკვეული წესით და შედეგად აგენერირებს პასუხს. წარმოვიდგინოთ რომ ნეირონში შედის ორი რიცხვი $x_1 = 2$ და $x_2 = 1.6$.

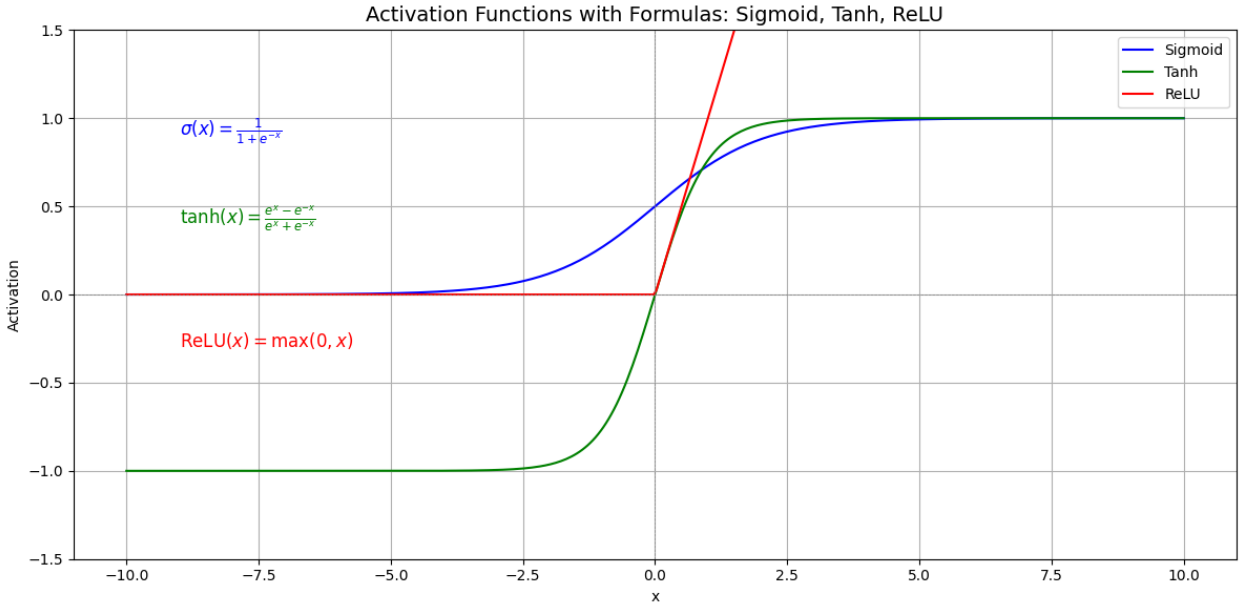


გრაფიკი 2.

ზემოთ ჩვენ ვახსენეთ პროცესი რომელშიც ითქვა რომ ნეირონი შემავალ მონაცემებზე ზემოქმედებს გარკვეული წესით, ეს წესი მდგომარეობს შემდეგში:

ყოველი X_i მრავლდება შესაბამის წონაზე W_i (i= [1, n]), ჯამდება და ემატება ეგრეთწოდებული წაძვრა b (bias). როდესაც ნეირონი ყენდება შრეზე, იგი დგება თავისი წინასწარ განსაზღვრული წონებით და წაძვრის მნიშვნელობით. ვთქვათ ზემოთ ნაჩვენები ნეირონის წონები და წაძვრა არის შემდეგი : W_1=1 , W_2=1 ხოლო b=0. აქედან გამომდინარე: 2*1+1.6*1+0=2+1.6+0=3.6. ნეირონმა გამოთვალა 3.6, რომელიც თითქმნის მზად არის გადაეცეს შემდეგ ნეირონს დასამუშავებლად იგივე პრინციპით. მაგრამ უნდა აღინიშნოს, რომ ნეირონებისთვის უფრო კომფორტულია პატარა მონაცემებთან მუშაობა, რადგან უფრო ზუსტად სრულდება გამოთვლები, ამიტომ ამ მიმდინარე ნეირონის მიზანია შემდეგ ნეირონს გადასცეს არა 3.6, არამედ უფრო მცირე რიცხვი, ასევე შემდეგმა ნეირონმა გადასცეს იმის შემდეგ ნეირონს ასევე მცირე რიცხვი და ასე ბოლომდე. ამისათვის შემოდის აქტივაციის ფუნქციების ცნება.

აქტივაციის ფუნქცია არის ფუნქცია, რომელიც მიღებულ რიცხვს „აპატარავებს“ შუალედში ან 0-დან 1-მდე, ან -1-დან 1-მდე ან 0-დან თვითონ გამოთვლილ პასუხამდე (გააჩნია ნეირონულ ქსელს და დავალების ტიპს). ასეთი ფუნქციებია მაგალითად : sigmoid, hyperbolic tanh, reLU.



გრაფიკი 3. ნეირონული ქსელის აქტივაციის ფუნქციები

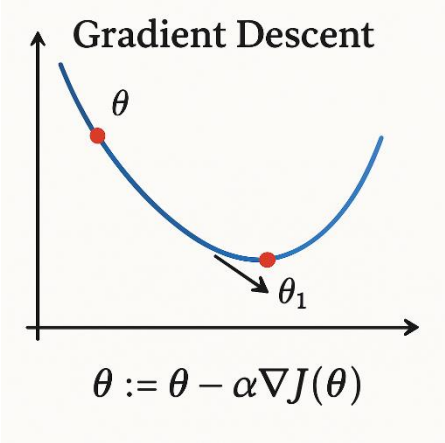
ამ შემთხვევაში წარმოვიდგინოთ, რომ გამოყენებულია სიგმოიდის აქტივაციის ფუნქცია და შედეგად გვექნება : output= $\frac{1}{1+e^{-3.6}}$, რაც დაახლოვებით არის 0.925. აქედან გამომდინარე ამ ნეირონის გამოსასვლელზე დაფიქსირდება 0.925. მაგრამ თუ ჩვენ ვართ ზედამხვედლებოთი სწავლების პროცესში, სადაც გვაქვს წინასწარი მონაცემები (dataset), რომელიც უნდა შევასწავლოთ ნეირონს, რათა შემდგომ თავად მოახდინოს კლასიფიკაცია და სიმაღლის და წონის მიხედვით დაადგინოს ქალია - 0 თუ კაცია -1 , პასუხი 0.925 არც თუ ისე სახარბიელოა, რადგან ნათლად არ ჩანს ეს კაცია თუ ქალი. ამიტომ ამ შემთხვევაში ფიქსირდება დანაკარგი Loss, ანუ ნეირონის შეცდომა კლასიფიკაციაში, რომელიც გამოითვლება ფორმულით:

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(Ypred_i - Ytrue_i\right)^2$$

სადაც n არის დაკვირვების რაოდენობა, dataset-ში სრეიქონების რაოდენობა , Ypred-ნეირონის მიერ ნაწინასწარმეტყველი მნიშვნელობა, Ytrue ნამდვილი მნიშვნელობა, მოცემული dataset-ში. აღწერილ პროცესს სწავლების დროს, უწოდებენ feedforward-ს.

სწავლების მიზანია ნეირონის წონები და წაძვრა შეცვალოს ისე, რომ ნაწინასწარმეტყველი პასუხი იყოს ნამდვილ პასუხთან ახლოს და დანაკარგის მნიშვნელობა უახლოვდებოდეს ნულს. იმისათვის რომ გავიგოთ თუ რამდენით უნდა შეიცვალოს თითოეული წონა და წაძვრა, დანაკარგის ფუნქცია უნდა გავაწარმოოთ W_1, W_2 და b მიხედვით და მოხდეს გაწარმოებული ფუნქციის ოპტიმიზაცია, ანუ მინიმუმის პოვნა. ამ პროცესს გრადიენტულ

დაშვებას უწოდებენ. თავის დაზღვევის მიზნით რათა მინიმუმის წერტილი არ იქნას გამორჩენილი, იყენებენ learning rate პარამეტრს, რომელიც ამცირებს დაშვების ბიჯებს.



გრაფიკი 4: გრადიენტული დაშვების გრაფიკი

მიღებული მნიშვნელობები ზუსტად მიგვითითებენ თუ როგორ გამოვთვალოთ ახალი წონები და წაძვრა. Wnew = wold-learning rate* derivative of L

ამ პროცესის გავლის შემდეგ სწავლების ციკლი ისევ უნდა განმეორდეს და იქნეს გათვალისწინებული ახლად გამოთვლილი წონები და წაძვრა რათა თავიდან შემოწმდეს დანაკარგის ფუქნციის მნიშვნელობა. თუ იგი ისევ დიდია, მაშინ მთლიანი პროცესი იწყება თავიდან და გრძელდება მანამ სანამ დანაკარგი არ მიუახლოვდება ნულს. ამ პროცესს back propagation-ს უწოდებენ. (Grossi , Buscema/გროსი, ბუსცემა. 2008)

1.2 კონვოლუციური ნეირონული ქსელი (CNN)

კონვოლუციური ნეირონული ქსელი (CNN — Convolutional Neural Network) არის ხელოვნური ნეირონული ქსელის სპეციალიზებული ტიპი, რომელიც განსაკუთრებით ეფექტურია სურათების, ვიდეოების და სივრცითი მონაცემების ამოცნობასა და ანალიზში. CNN გამოიყენება ბევრ თანამედროვე ამოცანაში: სურათის კლასიფიკაცია (ImageNet, CIFAR), ობიექტის ამოცნობა და ლოკალიზაცია, სახის ამოცნობა, სამედიცინო გამოსახულების ანალიზი და სხვ.

CNN ახდენს ავტომატურ მახასიათებლების ამოღებას სურათებიდან (Feature Extraction) კონვოლუციური ფილტრების საშუალებით.

CNN-ის ძირითადი ფენები:

1. კონვოლუციური ფენა (Convolutional Layer): იყენებს ფილტრს (kernel) — მცირე ზომის მატრიცას, რომელიც გადაადგილდება სურათზე. თითოეულ ადგილზე ის ითვლის დოტ-პროდუქტს (წერტილოვან ნამრავლს) და ქმნის feature map-ს. ამგვარად ამოიცნობა ფიგურები, ხაზები, კიდეები და სხვა ელემენტები.
2. აქტივაციის ფენა (Activation Layer): ჩვეულებრივ იყენებს ReLU ფუნქციას (Rectified Linear Unit): $f(x) = \max(0, x)$, არღვევს არარეგულარულობას და ქმნის არაწრფივობას.
3. პულინგის ფენა (Pooling Layer): ამცირებს სივრცობრივ განზომილებებს (height × width), ინარჩუნებს მნიშვნელოვან ინფორმაციას. მაგალითად, Max Pooling იღებს კონკრეტულ ფანჯარაში მაქსიმალურ მნიშვნელობას.
4. Flatten ფენა: გარდაქმნის 2D feature map-ებს ერთ განზომილებიან ვექტორად — მზად არის fully connected ფენისთვის.
5. Fully Connected Layer: ტრადიციული ნეირონული ფენა, რომელიც იყენებს ყველა წინა ამონათვალს და აგენერირებს საბოლოო კლასიფიკაციას ან რეგრესიას.

განვიხილოთ CNN-ის მუშაობის პრინციპი. როგორც ცნობილია, სურათის გრაფიკული გამოსახულება შედგება პიქსელებისგან და თითოეული პიქსელის შეფერილება მერყეობს 0-დან 255-მდე. ასეთი არხი გვაქვს სამი - RGB (Red, Green, Blue). იმისათვის რომ სურათის მახასიათებლები მარტივად და სწორად იქნეს ამოღებული კონვოლუციური ნეირონული ქსელის მიერ, საჭიროა ამ სურათის ტენზორის ნორმალიზაცია და მასშტაბირება. ამისათვის რამდენიმე ხერხი არსებობს, როგორიცაა StandardScaler, MinMaxScaler, ან თითოეული მნიშვნელობის 255-ზე გაყოფა. ამ პროცესს Image Preprocessing-ს უწოდებენ.

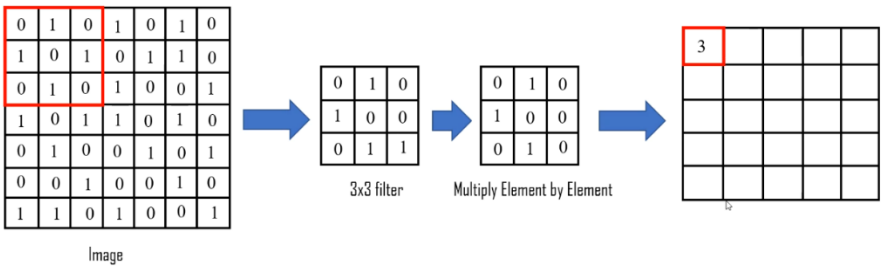
წარმოვიდგინოთ, რომ ეს პროცესი უკვე დასრულებულია და შედეგად გვაქვს შემდეგი სახის ნორმალიზებული ტენზორი (იხილეთ Figure 5):

0	1	0	1	0	1	0
1	0	1	0	1	1	0
0	1	0	1	0	0	1
1	0	1	1	0	1	0
0	1	0	0	1	0	1
0	0	1	0	0	1	0
1	1	0	1	0	0	1

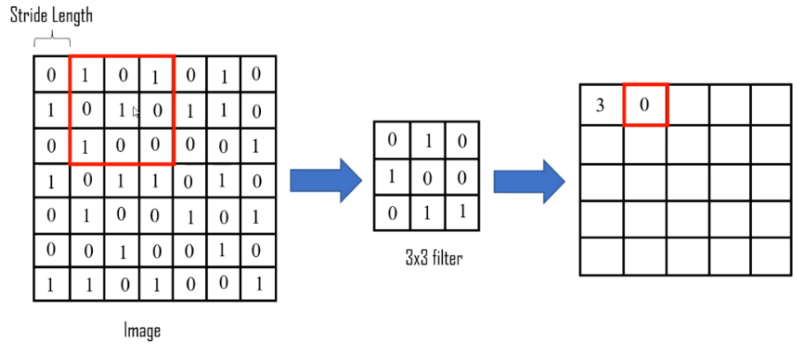
Image

გრაფიკი 5. გამოსახულების ნორმალიზებული ტენზორი

პირველ რიგში კონვოლუციური შრე ნორმალიზებულ ტენზორს ადებს ფილტრს. ფილტრი არის მატრიცა, რომელიც ცვლის სურათის ნორმალიზებულ ტენზორს იმისათვის, რომ აღმოაჩინოს მნიშვნელოვანი მახასიათებლები. ფილტრის მატრიცა შეიძლება იყოს ნებისმიერი ზომის, მაგრამ პრაქტიკაში ძირითადად გავრცელებულია ამ მატრიცის ზომა 3×3-ზე. ეს მატრიცა, თითო ბიჯით (Stride Length) გადაადგილდება ნორმალიზებულ ტენზორზე და თითო ელემენტის შესაბამისად გამრავლების შედეგად მიღებულ მნიშვნელობებს აგროვებს ახალ ტენზორში, რომელსაც Feature Map-ს უწოდებენ.

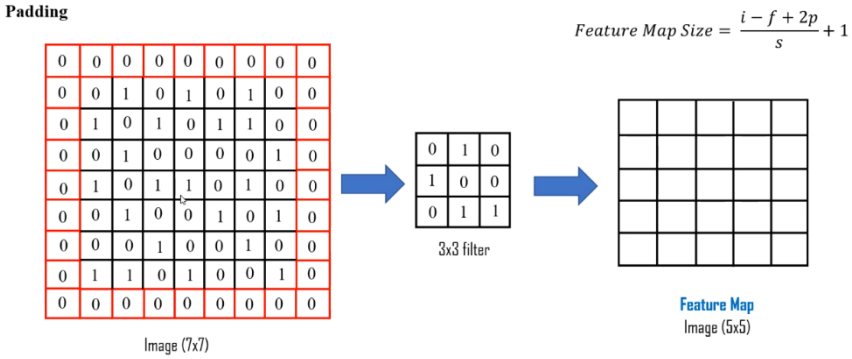


გრაფიკი 6. კონვოლუციური შრის სურათის ფილტრი



გრაფიკი 7. კონვოლუციური შრის სურათის ფილტრი

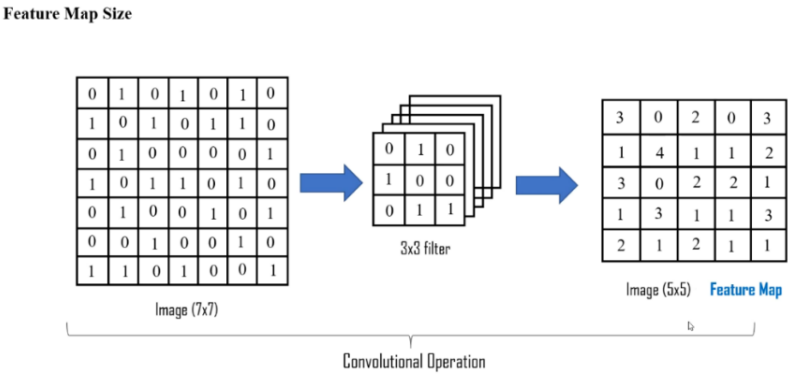
ასეთი ფილტრი შეიძლება ნებისმიერი რაოდენობის იყოს. ამ ოპერაციას კონვოლუციურ ოპერაციას უწოდებენ.



გრაფიკი 8. კონვოლუციური შრის სურათის ფილტრი

თუმცა ასეთი ოპერაციის დროს იქმნდება პრობლემა, რომელიც გულისხმობს შესაძლო მნიშვნელოვანი მახასიათებლების დაკარგვას. მაგალითად, თუ დავაკვირდებით ჩვენი სურათის ტენზორს და იმას თუ როგორ გადაადგილდება ფილტრი მასზე, შევნიშნავთ რომ პირველი სვეტის პირველ ელემენტს უფრო ნაკლები შანსი აქვს Feature Map-ში მოხვედრის ვიდრე სხვა დანარჩენს. ამიტომ ასეთი ნიუანსის გასათვალისწინებლად, დასახმარებლად შემოდის ცნება Padding. Padding სურათის ტენზორს ირგვლივ უმატებს

განზომილებას რომლებიც ნოლებით არის შევსებული და ამით ელემენტებს უმატებს შანსს, Feature Map-ში მოსახვედრად.



გრაფიკი 9. კონვოლუციური შრის სურათის ფილტრი

საინტერესოა როგორ განისაზღვრება Feature Map-ის ზომა პადინგით და პადინგის გარეშე. არსებობს ფორმულად რომლითაც შეგვიძლია ჩავატაროთ გამოთვლები.

პადინგის გარეშე : $\frac{i-f}{s} + 1$, სადაც i - ორიგინალი სურათის ტენზორის ზომაა , f - ფილტრის ზომაა , s - ფილტრის გადაადგილების ბიჯი (Stride Length). შედეგად : $\frac{7-3}{1} + 1 = 5$, ანუ Feature Map 5x5-ზე ტენზორია.

პადინგის არსებობის შემთხვევაში : $\frac{i-f+2p}{s} + 1$, სადაც i - ორიგინალი სურათის ტენზორის ზომაა, f - ფილტრის ზომაა, s - ფილტრის გადაადგილების ბიჯი (Stride Length), ხოლო p პადინგის რაოდენობა. ზემოთ ნაჩვენებ სურათებზე პადინგი ერთხელ არის გამოყენებული, ამიტომ $\frac{7-3+2*1}{1} + 1 = 7$, ამ შემთხვევაში Feature Map-ის ზომა იქნება 7x7-ზე.

შემდეგი ნაბიჯი არის Pooling, კონვოლუციურ ნეირონულ ქსელში (CNN – Convolutional Neural Network) არის ოპერაცია, რომელიც გამოიყენება ფილტრის მიერ გამოთვლილი მახასიათებლების ზომის შესამცირებლად და მოდელის გამოთვლითი ეფექტიანობის გასაზრდელად. Pooling ეხმარება ქსელს მნიშვნელოვანი ინფორმაციის შენარჩუნებაში და არარელევანტური დეტალების მოცილებაში.

Pooling-ის ძირითადი მიზნები:

1. გამოსახულების ზომის შემცირება – ამცირებს მონაცემების რაოდენობას, რაც ამარტივებს შემდეგ ფენებზე გამოთვლებს.
2. ზოგადი მახასიათებლების გამოკვეთა – ხელს უწყობს მნიშვნელოვანი დეტალების შენარჩუნებას და ხმის ან გადანაცვლების მიმართ მედეგობას.
3. გადატრენინგების თავიდან არიდება – მონაცემების ზომის შემცირებით, კომპლექსურობა მცირდება, რაც ეხმარება რეგულარიზაციაში.

Pooling-ის ტიპები:

1. Max Pooling – იღებს მაქსიმუმს მცირე რეგიონიდან (მაგ., 2x2 ფანჯრიდან).

   მაგალითად:

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \Rightarrow \max(4)$$

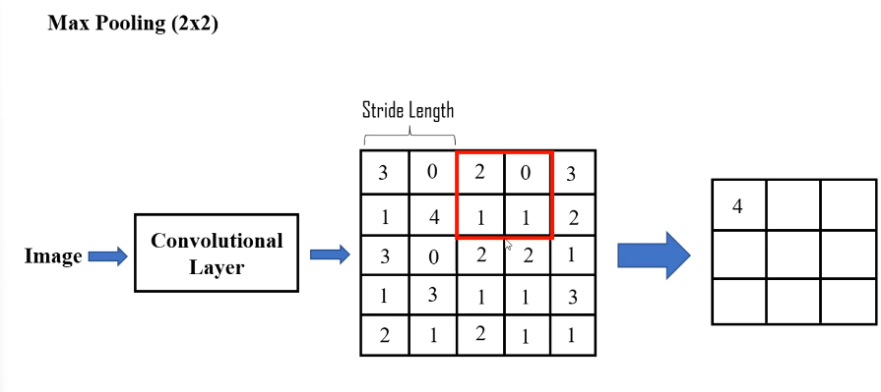
2. Average Pooling – იღებს საშუალოს იმავე რეგიონიდან.

   იგივე მაგალითში:

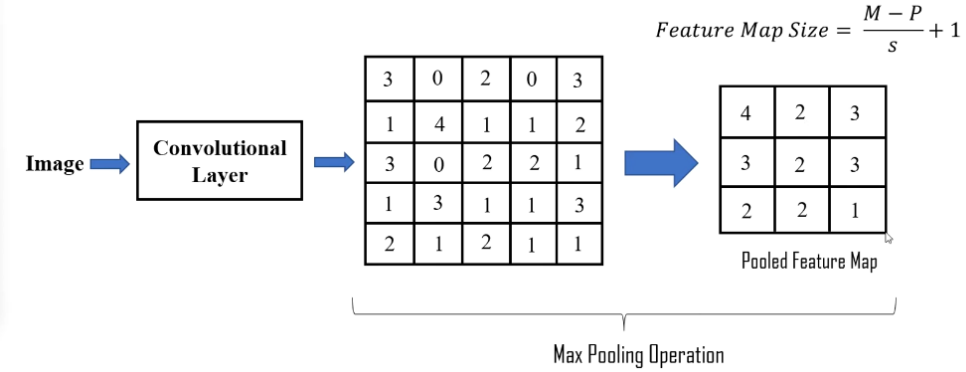
$\frac{1+2+3+4}{4}$ =2.5

როგორ მუშაობს: Pooling ფანჯარა მოძრაობს მახასიათებელ რუკაზე (feature map), ჩვეულებრივ სტრაიდით (მაგ., 2) და ყოველ ნაბიჯზე ასრულებს განსაზღვრულ pooling ოპერაციას. შედეგი არის შემცირებული ზომის რუკა, რომელიც კვლავ ინარჩუნებს ყველაზე მნიშვნელოვან ინფორმაციას.

როგორც ვატყობთ MAX Pooling ოპერაცია უფრო აგრესიულია ვიდრე Average Pooling, ამიტომ გავრცელებული პრაქტიკაა MaxPooling გამოყენება.



გრაფიკი 10. გაერთიანება (Pooling)

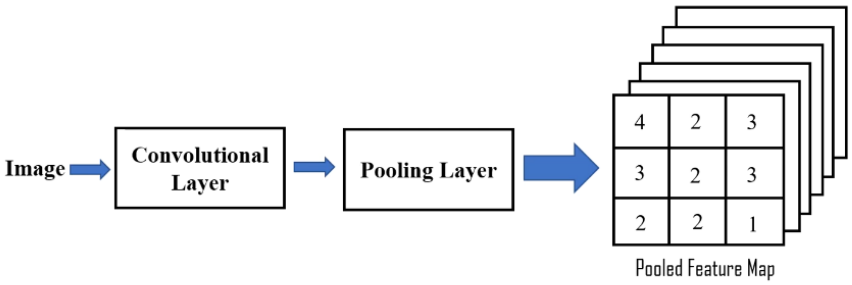


გრაფიკი 11. გაერთიანება (Pooling)

შედეგად შექმნილი Pooled feature map-ის ზომის გამოთვლაც შესაძლებელია წინასწარ ფორმულით, სადაც M- Feature Map ზომაა, P- Pooling ფილტრის მატრიცის ზომა და s – Stride Length, ბიჯი რომლითაც გადადგილდება pooling-ის ფილტრი Feature map-ზე. აქედან გამომდინარე თუ ვიტყვით რომ Feature Map იყო 5x5-ზე ,

pooling-ის ფილტრის მატრიცის ზომა 2x2-ზე ხოლო ბიჯი (Stride Length) არის 2, მაშინ გვექნება: $\frac{5-2}{2} + 1 = 2.5$, დამრგვალების წესით გამოითვლება 3, რაც ნიშნავს იმას რომ Pooled feature map მატრიცის ზომა არის 3x3-ზე.

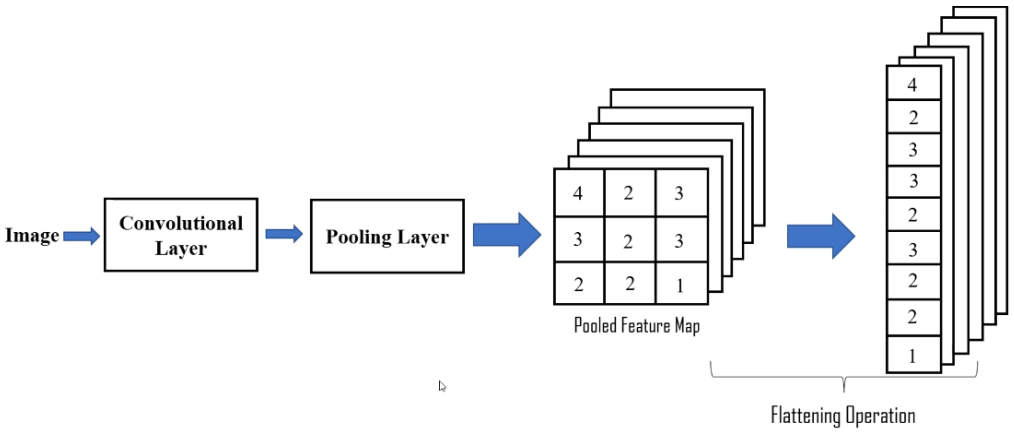
კონვოლუციური ნეირონული ქსელის ბოლო ნაბიჯი არის Flattening. Flattening



გრაფიკი 12. გაბრტყელება (flattening)

კონვოლუციურ ნეირონულ ქსელში (CNN) არის პროცესი, რომელიც გამოიყენება მახასიათებლების რუკის (feature map) მრავალგანზომილებიანი სტრუქტურის გარდასაქმნელად ერთგანზომილებიან ვექტორად. CNN-ის საწყისი ფენები (კონვოლუციური და pooling ფენები) ამუშავებს მონაცემებს 2D ან 3D ფორმატში (მაგალითად, 224x224x3). მაგრამ ნეირონული ქსელის ბოლო ნაწილში (ე.წ. Fully Connected ფენებში), საჭიროა ერთგანზომილებიანი ვექტორი, რადგან ამ ფენებში ყოველი ნეირონი უნდა იყოს დაკავშირებული წინა ფენის თითოეულ გამოსასვლელთან. Flatten გამოიყენება პირდაპირ pooling-ის ან ბოლო კონვოლუციური ფენის შემდეგ, სანამ მონაცემები მიეწოდება:

1. Dense (Fully Connected) ფენებს
2. კლასიფიკაციისთვის (Softmax/Logits ფენამდე)



გრაფიკი 13. გაბრტყელება (Flattening)

ამ კოდის სნიპიეტში ნაჩვენებია კონვოლუციური ნეირონული ქსელის არქიტექტურა PyTorch Framework-ზე, სადაც მუშავდება სურათი, რომელიც ნორმალიზებულია ზომამდე 100×100-ზე და აქვს 3 არხი RGB, ანუ არის ფერადი, ამიტომ შესასვლელი არის 3, დაყენებულია 50 ფილტრი ზომით 3×3-ზე (kernel size), რომლებიც გადადგილდებიან ერთი ბიჯით (Stride Length) და დაყენებულია ერთი ცალი Padding. შემდეგ ოპერაციას აგრძელებს MaxPooling-ი, რომლის ზომაც არის 2×2-ზე (kernel size) და გადაადგილდებაორი ბიჯით (Stride Length). პირველი შრის შემდეგ გამოითვლება შემდეგი კონვოლუციური შრის შესასვლელის ზომა და გამოსასვლელის ზომა Fully Connected შრისთვის. ასევე ხდება Flattening ოპერაცია Forward მეთოდში მანამ სანამ გადაეცემა Fully Connected შრეს.

```
class CNN(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv=nn.Sequential(
          nn.Conv2d(in_channels=3,out_channels=50,kernel_size=3,stride=1,padding=1),     #(100-3+2*1)/1 + 1= 100    M=(i-f+2p)/s + 1
        nn.MaxPool2d(kernel_size=2,stride=2), #() (M-P)/S +1 (100-2)/2 +1=50
        nn.ReLU(),
```

```
            nn.Conv2d(in_channels=50,out_channels=30,kernel_size=3,stride=1,padding=1),    #(50-3+2*1)/1 +1 = 50
        nn.MaxPool2d(kernel_size=2,stride=2), #(50-2)/2 +1 = 25
        nn.ReLU()
    )
    self.fc=nn.Sequential(
                nn.Linear(in_features=30*25*25,out_features=512), #ბოლო კონვოლუციურის გამოსასვლელის რაოდენობა გამრავლებული ბოლოს მაქსპულინგის გამოსასვლელის ზომის კვადრატზე
        nn.ReLU(),
        nn.Linear(in_features=512,out_features=256),
        nn.ReLU(),
        nn.Linear(256,128),
        nn.ReLU(),
        nn.Linear(128,50)
    )
  def forward(self,x):
    x=self.conv(x)
    x=x.view(x.size(0),-1) #Flattening ოპერაცია
    x=self.fc(x)
    return x
```

(Kulkarni,Premraj,Ordonez,Dhar,Li,Choi, Berg,Berg/კულკარნი, პრემრაჟი,ორდონეზი,დარი,ლი,ჩოი. 2013)

1.3 რეკურენტული ნეირონული ქსელი და Long-short Term მეხსიერება

რეკურენტული ნეირონული ქსელები (RNN, Recurrent Neural Networks) და მათი გაუმჯობესებული ვერსია — LSTM (Long Short-Term Memory) — სპეციალურად შექმნილია დროში დამოკიდებული მონაცემების დასამუშავებლად, როგორიცაა: ტექსტი, ხმა, დროის სერიები, მუსიკა.

RNN — ხელოვნური ნეირონული ქსელის ტიპია, რომელიც განსაკუთრებით ეფექტურია მიმდევრობითი (sequence-based) მონაცემების დამუშავებისთვის. ანუ მონაცემებისთვის, სადაც წარსული მნიშვნელობა გავლენას ახდენს მომავალზე.

ტრადიციული ნეირონული ქსელები არ იმახსოვრებენ წინა ინფორმაციის კონტექსტს. RNN კი "მეხსიერების" მექანიზმით მუშაობს: ის ინახავს წარსულ მდგომარეობას და ამ მდგომარეობის საფუძველზე იღებს გადაწყვეტილებებს შემდეგ ეტაპზე.

RNN-ები გამოიყენება ისეთ ამოცანებზე, სადაც მონაცემებს თანმიმდევრობა აქვთ:

1. ტექსტის გენერაცია (მაგ. ტექსტის გაგრძელება)
2. ნატურალური ენის დამუშავება (NLP) — ტექსტის კლასიფიკაცია, ენების მოდელირება, თარგმნა
3. დროითი სერიის პროგნოზი (მაგ. საფონდო ბაზრის პროგნოზირება)
4. ხმოვანი სიგნალების ამოცნობა (მაგ. აუდიოდან ტექსტში გადაყვანა — speech recognition)
5. მუსიკის გენერაცია

თუმცა ამ ტიპის ნეირინულ ქსელს აქვს შემდეგი პრობლემები:

1. Vanishing Gradient: Back Propagation-ის დროს, როცა ნეირონული ქსელი სწავლობს, მნიშვნელოვანი გრადიენტების ნელ-ნელ ქრებიან, ამის შედეგად, RNN ვერ "იმახსოვრებს" ძველი მნიშვნელობების გავლენას — რაც უარყოფითად ისახება გრძელვადიან დამოკიდებულებებზე (long-term dependencies).
2. Exploding Gradient: ზოგჯერ გრადიენტები Back Propagation-ის დროს, ძალიან დიდდება, რაც ქსელის სასწავლ პროცესს არასტაბილურს ხდის.
3. გრძელვადიანი დამოკიდებულებების სირთულე: RNN-ებს უჭირთ შორეული წარსული მნიშვნელობების "დამახსოვრება". მაგალითად, ტექსტში პირველი წინადადების გავლენა შეიძლება დაიკარგოს მეხუთე წინადადებამდე მისვლისას.

LSTM (Long Short-Term Memory) — სპეციალური ტიპის RNN-ია, რომელიც შეიქმნა სწორედ იმისთვის, რომ გრძელვადიანი დამოკიდებულებების დამახსოვრება უკეთესად შეძლოს.

LSTM-მა გადაჭრა vanishing gradient-ის პრობლემა gating mechanism-ის საშუალებით

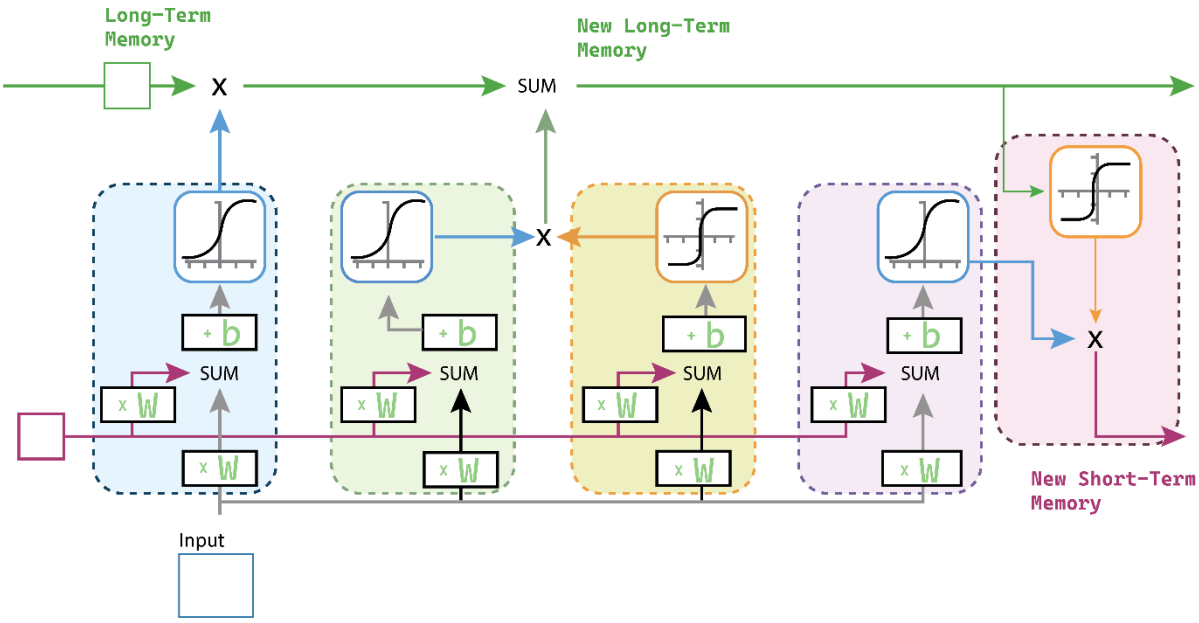
LSTM იყენებს სამ ძირითად კარს:

1. Forget gate — რა დავივიწყოთ?
2. Input gate — რა დავამატოთ მეხსიერებაში?
3. Output gate — რა გამოვიყენოთ მეხსიერებიდან?

ამ კარების წყალობით, LSTM სწავლობს, რომელი ინფორმაციაა მნიშვნელოვანი დასამახსოვრებლად და რომელი — არა.

LSTM თითქმის იგივე ამოცანებზე გამოიყენება, როგორც RNN, მაგრამ უკეთეს შედეგებს იძლევა:

1. ტექსტის თარგმნა (მაგ. Google Translate)
2. ჩათბოტები (მაგ. მობაილ ასისტენტები)
3. მუსიკის ან ტექსტის გენერაცია
4. დროითი სერიის პროგნოზი
5. ბიომეტრიული მონაცემების ანალიზი (მაგ. ECG სიგნალის გაშიფვრა)



გრაფიკი 14. LSTM კვანძი

განვიხილოთ LSTM უჯრედის არქიტექტურა და მუშაობის პრინციპი

ნახაზს თუ დავაკვირდებით შევამჩნევთ ნაკადს, რომელიც მწვანე ფერით არის აღნიშნული და ნაკადს რომელიც იასამნისფრად არის აღნიშნული. მწვანე ფერით აღნიშნულ ნაკადს Cell State-ს ანუ უჯრედის მდგომარეობას უწოდებენ, ხოლო იასამნისფრად აღნიშნულ ნაკადს Hidden State ანუ უჯრედის დამალულ მდგომარეობას. პირველი ემსახურება იმას თუ რა ინფორმაცია უნდა იქნას დამახსოვრებული გრძელვადიანად, ხოლო მეორე ეგრეთწოდებული მუშა ნაკადია.

Cell State-ში არის მოცემული ორი ოპერაცია : გამრავლება და ჯამი. ის მონაკვეთი სადაც ხდება გამრავლება უწოდებენ Forget Gate-ს, ანუ ეს არის ადგილი, სადაც განისაზღვრება რა უნდა იქნას დამახსოვრებული უჯრედის მიერ და რა არა, ხოლო დაჯამების მონაკვეთს Input Gate-ს უწოდებენ, სადაც განისაზღვრება თუ კიდევ რა უნდა იქნას ჩამატებული მეხსიერებაში დასამახსოვრებლად. სულ ბოლო კვანძს კი Output Gate ეწოდება, საიდანაც გამოიცემა ინფორმაცია შემდეგი LSTM უჯრედისთვის. აღსანიშნავია რომ კვანძში ორი აქტივაციის ფუნქცია მუშაობს, ესენია : სიგმოიდი და ჰიპერბოლური ტანგენსი. Forget Gate-ში ზუსტად სიგმოიდის აქტივაცის ფუნქცია დგას, რადგან იგი ყოველთვის იძლევა პასუხს ან 0 ან 1, რაც იწვევს იმას რომ გამრავლების შედეგად 0-ზე ყველაფერი ნულია და დავიწყებას მიეცემა, ხოლო 1 გამრავლებისას განაგრძობს გზას მეხსირებაში. რაც შეეხება Input Gate-ს სადაც უკვე ორი აქტივაციის ფუნქცია მუშაობს, სიგმოიდთან ერთად ჰიპერბოლური ტანგენსი, ერთი განსაზღვრავს პოტენციურ დასამახსოვრებელ მნიშვნელობას (ჰიპერბოლური ტანგენსი [-1,1]) , ხოლო სიგმოიდი იღებს გადაწყვეტილებას დაემატოს ეს მნიშვნელობა მეხსირებაში თუ არა, გადამრავლების ოპერაციის შედეგად. ბოლო ეტაპზე (Output Gate), მეხსირებაში არსებული მნიშვმელობა და „მუშა“ ნაკადში არსებული მნიშვნელობა სიგმოიდის და ჰიპერბოლური ტანგენსის ფუქნციის გავლით ისევ განიცდის გადამრავლების ოპერაციას, რის შედეგადაც განისაზღვრება ამ ნაკადის გამოსასვლელის პასუხი, რომელიც გრძელვადიანი მეხსიერების მნიშვნელობასთან ერთად გადაეცემა შემდეგ LSTM უჯრედს.

ცხრილი 1. რეკურენტული ნეირონული ქსელის და LSTM ნეირონული ქსელის შედარებითი ანალიზი

საკითხი	RNN	LSTM

გრძელვადიანი მეხსიერება	შეზღუდულია	გაუმჯობესებული
გრადიენტის პრობლემები	ხშირია (vanishing/exploding)	მნიშვნელოვნად შემცირებულია
გამოყენება	NLP, დროითი სერიები	იგივე, მაგრამ უკეთესი შედეგით

ამ ორი ტიპის ნეირონული ქსელის საშუალებით იქნებოდა შესაძლებელი ისეთი ხელოვნური ინტელექტის შექმნა, რომელიც გაანალიზებდა გამოსახულებას და შემდგომ აღწერდა დანახულს სიტყვიერად. ასეთი AI არსებობს, თუმცა საერთოდ სხვა მიდგომებს იყენებს აღნიშნული დავალებისთვის. იგი შექმნილია თავიდან ბოლომდე transformer ტექნოლოგიით, რადგან CNN-ები სწავლობენ ლოკალურ თვისებებს (ანუ კუთხეებს, ტექსტურას). კონვოლუციური ფილტრები მხოლოდ ახლო მეზობლებს ხედავენ, ხოლო ViT-ები (Vision Transformers) იყენებენ თვითყურადღების მექანიზმს (self-attention). ეს გარემოება საშუალებას აძლევს თითოეულ "პატჩს" (patch), პატარა N×N პიქსელიანი ბლოკი, რომელსაც ViT სურათიდან ჭრის და მასთან მიმართებით სწავლობს, გაითვალისწინოს მთლიან სურათში არსებული შორეული ნაწილებიც. ეს განსაკუთრებით მნიშვნელოვანია ობიექტის გლობალური სტრუქტურისთვის. ტრანსფორმერები უკეთესად ითვისებენ ისეთ დამოკიდებულებებს, რაც შეიძლება CNN-ებს გაუჭირდეს — მაგალითად, როდესაც ობიექტის ნაწილები ერთმანეთისგან შორსაა განთავსებული.ViT უკეთესად სწავლობს დიდ მონაცემებზე. Google-ის კვლევებმა აჩვენა, რომ როდესაც საკმარისი მონაცემია და გამოთვლითი რესურსი არსებობს, ViT ბევრად სჯობნის CNN-ს.

იგივე ითქმის LSTM-ზე, BERT ტრანსფორმერთან შედარების დროს

ცხრილი 2. BERT და LSTM შედარება

თვისება	BERT	LSTM
არქიტექტურა	ტრანსფორმერი (Transformer, encoder-only)	რეკურენტული ნეირონული ბადე (RNN)
პარალელიზაცია	სრულად პარალელიზებადია	სერიული — მუშაობს

		თანმიმდევრულად
დამოკიდებულებების დამუშავება	ხედავს მთელ წინადადებას ერთდროულად (bidirectional)	წინ ან უკან (unidirectional / bidirectional, მაგრამ მაინც თანმიმდევრულად)
პრეტრენინგი	ჩვეულებრივ წინასწარ გაწვრთნილია დიდ კორპუსებზე (e.g. Wikipedia)	ხშირად ნულიდან წვრთნიან
შესრულება	უმეტესად უკეთესია NLP ამოცანებზე	შედარებით სუსტია კომპლექსურ ამოცანებში

(Donahue,Hendricks,Guadarrama,Rohrbach,Venugopalan, Saenko, Darrell/დონაჰუ,ჰენფრიკსი,გუადარამა,როჰბრახი,ვენუგოპალანი,საენკო, დარელი. 2015),

(Young,Yuan,Wu, Salakhutdinov,Cohen/იანგი,იუანი,ვუ,სალახუტდინოვი. 2016)

(Huang,Rathod,Sun, Zhu, Korattikara,Fathi,Fischer,Wojna,Song,Guadarrama ,Murphy/ჰუანგი,რათოდი,სან, ჟუ, კორატიკარა, ფატი,ფიშჩერი,ვოჟნა,სონგი,გუადარმა, მერფი.2017)

(Ren,He,Girshick,Sun/რენ,ჰე,გირშიკ,სან. 2015)

თავი II

Blip-2 , ByT5 და Martha მოდელის მათემატიკური მოდელირება

ხელოვნური ინტელექტის სწრაფი განვითარება ბოლო ათწლეულში ახალ შესაძლებლობებს ქმნის როგორც მეცნიერებაში, ისე ინდუსტრიულ სექტორებში. კომპიუტერული ხედვისა და ბუნებრივი ენის დამუშავების (Computer Vision & NLP) შერწყმა ერთ-ერთი ყველაზე აქტუალური მიმართულებაა, რომელიც საშუალებას იძლევა ვიზუალური ინფორმაციის ავტომატური ინტერპრეტაცია განხორციელდეს ტექსტურ ფორმაში. ამ ტიპის სისტემები უკვე ფართოდ გამოიყენება ავტონომიურ ტრანსპორტში, რობოტიკაში, მედიცინაში, უსაფრთხოებაში და ინკლუზიური ტექნოლოგიების სფეროში.

თუმცა, გლობალური AI მოდელების უმეტესობა ინგლისურ ენაზეა ორიენტირებული. მცირე ენები, მათ შორის ქართული, ხშირად რჩება ტექნოლოგიური განვითარების მიღმა. ქართული ენისთვის სპეციფიკური გამოწვევებია:

1. კომპლექსური მორფოლოგია,
2. მონაცემთა ნაკლებობა,
3. არასტანდარტული ორთოგრაფიული ნიუანსები,
4. მრავალფეროვანი სტილისტური და ლექსიკური ფორმები.

ამ კონტექსტში პროექტი „Martha“ მიზნად ისახავს ვიზუალური გარემოს აღწერის სისტემის შექმნას ქართულ ენაზე, რომელიც დაფუძნებულია თანამედროვე Encoder–Decoder არქიტექტურაზე. „Martha“-ში ინტეგრირებულია ორი ძლიერი კომპონენტი: BLIP2 ვიზუალური encoder-ისთვის და ByT5 ტექსტის გენერაციისთვის.

პროექტის შედეგი იქნება:

1. ქართულ ენაზე ფუნქციონირებადი კომპიუტერული ხედვის მოდელი,
2. ინკლუზიური ტექნოლოგია მხედველობადაქვეითებულთათვის,
3. მონაცემთა ღია რესურსი (Georgian Image-Caption Dataset),
4. მეცნიერული სარგებელი როგორც ეროვნულ, ისე საერთაშორისო დონეზე.

2.1 Encoder–Decoder არქიტექტურა

Encoder–Decoder მოდელები მრავალი წელია წარმატებით გამოიყენება სურათიდან ტექსტის გენერაციაში. Encoder ახორციელებს შეყვანილი სურათის სემანტიკურ წარმოდგენას, ხოლო Decoder ამ ინფორმაციას გარდაქმნის ტექსტურ გამოსავალში.

ზოგადი ფორმულირება:

$$z=f\theta(x), y=g\phi(z)$$

სადაც x — შეყვანილი სურათია, z — სემანტიკური ვექტორი, $f\theta$— encoder, ხოლო $g\phi$ — decoder.

2.2 BLIP2

BLIP2 (Bootstrapping Language-Image Pretraining) წარმოადგენს მოდელს, რომელიც გაწვრთნილია ასობით მილიონი სურათ–ტექსტის წყვილზე. მისი მთავარი ინოვაციაა Q-Former – Query Transformer, რომელიც სურათიდან აღებულ ინფორმაციას ეფექტურად აკომპაქტებს ენობრივი მოდელისთვის მისაცემად. (Zhu,Chen,Haydarov,Shen,Zhang,Elhoseiny/ჟუ,ჩენ,ჰაიდაროვი,შენ,ჟანგ,ელჰოსეინი. 2023)

ვიზუალურ-ენობრივი წინასწარი სწავლება (Vision-Language Pre-training, VLP) ბოლო წლებში სწრაფად განვითარდა, რასაც ხელი შეუწყო სულ უფრო მასშტაბურმა მულტიმოდალურმა მოდელებმა, რომლებმაც მაღალი შედეგები აჩვენეს მრავალფეროვან ქვედა დონის ამოცანებზე, როგორიცაა ვიზუალურ კითხვებზე პასუხი (VQA), კაპშენინგი და რეტრივალი (Radford,Kim,Hallacy,Ramesh,Goh,Agarwal/რადფორდი,კიმი,ჰალასი,რამეში,გოჰი,აგერვალი. 2021; Li,Selvaraju,Gotmare,Joty,Xiong,Hoi/ლი,სელვარაჯუ,გოტმარე,ჯოტი,სიონგი,ჰოი. 2021; Li,Li,Xiong,Hoi/ლი,ლი,სიონგი,ჰოი. 2022; Wang,Yang,Men,Lin,Bai,Li/ვანგი,იანგი,მენი,ლინი,ბაი,ლი. 2022a; Alayrac,Donahue,Luc,Miech,Barr,Hasson/ალაირაკი,დონაჰიუ,ლუკი,მიეში,ბარი,ჰასონი. 2022; Wang,Bao,Dong,Bjorck,Peng,Liu/ვანგი,ბაო,დონგი,ბიორკი,პენგი,ლიუ. 2022b). თუმცა, თანამედროვე მოწინავე მიდგომების დიდი ნაწილი ეყრდნობა დიდი მასშტაბის არქიტექტურების end-to-end სწავლებას უზარმაზარ მონაცემთა ბაზებზე, რაც იწვევს მნიშვნელოვან გამოთვლით ხარჯს და ამცირებს მოქნილობას. BLIP-2 ამ პრობლემას სთავაზობს გამოთვლითად ეფექტიან ალტერნატივას, რომელიც იყენებს „გაყინულ“ (frozen) წინასწარ გაწვრთნილ უნიმოდალურ მოდელებს: ვიზუალური და ენობრივი კომპონენტების თავიდან ერთობლივი სწავლების ნაცვლად, ჩარჩო შემოაქვს მსუბუქ Querying Transformer-ს (Q-Former), რომელიც აკავშირებს გაყინულ გამოსახულების ენკოდერს გაყინულ დიდ ენობრივ მოდელთან (LLM); ეს მოდულური სტრატეგია მკვეთრად ამცირებს სასწავლო (trainable) პარამეტრების რაოდენობას და თან ინარჩუნებს კონკურენტულ ან უფრო მაღალ შესრულებას. BLIP-2-ის მთავარი ამოცანა არის მოდალობების გასწორება (modality alignment): რადგან LLM-ები წმინდად ტექსტურ მონაცემებზე ისწავლება, მათ არ აქვთ შინაგანი ვიზუალური „დამიწება“ (visual grounding), ხოლო LLM-ის გაყინვა ჯვარედინი-მოდალური გასწორების ამოცანას კიდევ უფრო

ართულებს; BLIP-2 ამას აგვარებს ორსაფეხურიანი წინასწარი სწავლებით, რომელიც ეტაპობრივად აკავშირებს ვიზუალურ და ტექსტურ წარმოდგენებს. არსებული VLP მეთოდები იყენებს სხვადასხვა არქიტექტურულ პარადიგმას, მათ შორის dual-encoder მოდელებს (Radford,Kim,Hallacy,Ramesh,Goh,Agarwal/რადფორდი,კიმი,ჰალასი,რამეში,გოჰი,აგერვალი . 2021; Jia,Yang,Xia,Chen,Parekh,Pham/ჯია,იანგი,სია,ჩენი,პარეხი,ფამი. 2021), fusion encoder-ებს (Tan,Bansal/ტანი,ბანსალი. 2019; Li,Selvaraju,Gotmare,Joty,Xiong,Hoi/ლი,სელვარაჯუ,გოტმარე,ჯოტი,სიონგი,ჰოი. 2021), encoder-decoder სტრუქტურებს (Cho,Lei,Tan,Bansal/ჩო,ლეი,ტანი,ბანსალი. 2021; Wang,Yu,Yu,Dai,Tsvetkov,Cao/ვანგი,იუ,იუ,დაი,ცვეტკოვი,კაო. 2021b; Chen,Wang,Changpinyo,Piergiovanni,Padlewski,Salz/ჩენ,ვანგი,ჩანგპინიო,პიერჯიოვანი,პედ ლევსკი,სალცი. 2022b) და unified transformer ჩარჩოებს (Li,Li,Xiong,Hoi/ლი,ლი,სიონგი,ჰოი. 2022; Wang,Bao,Dong,Bjorck,Peng,Liu/ვანგი,ბაო,დონგი,ბიორკი,პენგი,ლიუ. 2022b). გავრცელებულ მიზნებს შორისაა image-text კონტრასტული სწავლება, image-text matching და masked language modeling; მიუხედავად ეფექტიანობისა, ეს მიდგომები მოითხოვს მოდალობებს შორის ძვირადღირებულ ერთობლივ ოპტიმიზაციას. ბოლო კვლევები ცდილობს ტრენინგის ხარჯის შემცირებას უნიმოდალური კომპონენტების გაყინვით: LiT (Zhai,Wang,Mustafa,Steiner,Keysers,Kolesnikov/ჟაი,ვანგი,მუსტაფა,სტაინერი,კეიზერსი,კოლ ესნიკოვი. 2022) ყინავს image encoder-ებს, ხოლო Flamingo (Alayrac,Donahue,Luc,Miech,Barr,Hasson/ალაირაკი,დონაჰიუ,ლუკი,მიეში,ბარი,ჰასონი. 2022) ყინავს LLM-ს და ვიზუალურ სიგნალებს შეჰყავს cross-attention ფენებით; მიუხედავად ამისა, გაყინული მოდელების გასწორება კვლავ რთულია. BLIP-2-ის განმასხვავებელი ნიშანია ის, რომ იგი ყინავს როგორც ვიზუალურ, ისე ენობრივ „ბექბოუნებს“ და ამავე დროს ამატებს გაწვრთნად შუამავალ ტრანსფორმერს. Q-Former არის ერთადერთი გაწვრთნადი მოდული, რომელიც აკავშირებს გაყინულ image encoder-ს (მაგ., CLIP-ის ViT-L/14 (Radford,Kim,Hallacy,Ramesh,Goh,Agarwal/რადფორდი,კიმი,ჰალასი,რამეში,გოჰი,აგერვალი . 2021) ან EVA-CLIP-ის ViT-g/14 (Fang,Wang,Xie,Sun,Wu,Wang/ფანგი,ვანგი,სიე,სანი,უ,ვანგი. 2022)) გაყინულ LLM-თან (მაგ., OPT (Zhang,Roller,Goyal,Artetxe,Chen,Chen/ჟანგი,როლერი,გოიალი,არტეტქსე,ჩენ,ჩენ. 2022) ან FlanT5 (Chung,Hou,Longpre,Zoph,Tay,Fedus/ჩუნგი,ჰოუ,ლონგპრე,ზოფი,ტეი,ფედუსი.

2022)); იგი შედგება ტრანსფორმერის ბლოკებისგან, რომლებიც იზიარებს self-attention ფენებს და ამატებს cross-attention ფენებს ვიზუალური ნიშნების გამოსატანად. ფიქსირებული რაოდენობის სასწავლო query-ემბედინგები cross-attention-ის საშუალებით ურთიერთქმედებს image feature-ებთან და მაღალი განზომილების ვიზუალურ წარმოდგენებს შეკუმშავს უფრო მცირე, ამოცანაზე რელევანტურ ემბედინგებად; ეს „ბოთლნექი“ აიძულებს მოდელს შეინარჩუნოს მხოლოდ ის ვიზუალური ინფორმაცია, რომელიც ტექსტურ სემანტიკასთანაა გასწორებული. პირველ საფეხურზე Q-Former იწვრთნება გაყინულ image encoder-თან ერთად სამი დამატებითი მიზნით: image-text contrastive learning (ITC), image-text matching (ITM) და image-grounded text generation (ITG), სადაც განსხვავებული attention mask-ები არეგულირებს query-ებისა და ტექსტური ტოკენების ურთიერთქმედებას; ITC კონტрастული დანაკარგით ასწორებს ვიზუალურ query-წარმოდგენებს ტექსტურ ემბედინგებთან, ITM ბინარული კლასიფიკაციით ამყარებს წყვილთა (match/mismatch) წვრილმარცვლოვან მულტიმოდალურ გასწორებას, ხოლო ITG ვიზუალურ ნიშნებზე პირობებულ ტექსტის გენერაციას მოითხოვს და ხელს უწყობს ენობრივად რელევანტური ვიზუალური ინფორმაციის გამოყოფას. მეორე საფეხურზე Q-Former ერთდება გაყინულ LLM-თან ლინეარული პროექციის ფენით: პროექცირებული query-ემბედინგები თავსდება ტექსტური შეყვანის წინ და მოქმედებს როგორც „რბილი“ ვიზუალური პრომპტები; decoder-only LLM-ებისთვის გამოიყენება სტანდარტული language modeling დანაკარგი, ხოლო encoder-decoder LLM-ებისთვის—prefix language modeling, სადაც მიმდევრობის ნაწილი პირობად უდგას გენერაციას. BLIP-2 წინასწარ იწვრთნება 129 მილიონ image-text წყვილზე COCO-დან (Lin,Maire,Belongie,Hays,Perona,Ramanan/ლინი,მაერე,ბელონჟი,ჰეიზი,პერონა,რამანანი. 2014), Visual Genome-დან (Krishna,Zhu,Groth,Johnson,Hata,Kravitz/კრიშნა,ჟუ,გროთი,ჯონსონი,ჰატა,კრავიცი. 2017), CC3M-იდან (Sharma,Ding,Goodman,Soricut/შარმა,დინგი,გუდმენი,სორიქუტი. 2018), CC12M-იდან (Changpinyo,Sharma,Ding,Soricut/ჩანგპინიო,შარმა,დინგი,სორიქუტი. 2021), SBU-დან (Ordonez,Kulkarni,Berg/ორდონესი,კულკარნი,ბერგი. 2011) და LAION400M-იდან (Schuhmann,Vencu,Beaumont,Kaczmarczyk,Mullis,Katta/შუხმანი,ვენკუ,ბომონი,კაჩმარიჩკი, მალისი,კატა. 2021); სინთეტიკური კაპშენები გენერირდება CapFilt მეთოდით (Li,Li,Xiong,Hoi/ლი,ლი,სიონგი,ჰოი. 2022). ნულ-შოთ VQA ექსპერიმენტები აჩვენებს, რომ BLIP-2 VQAv2-ზე აჭარბებს Flamingo80B-ს მნიშვნელოვნად ნაკლები trainable პარამეტრებით, ხოლო შედეგები უმჯობესდება უფრო ძლიერი image encoder-ებისა და

უფრო დიდი LLM-ების გამოყენებით; image captioning-ში BLIP-2 აღწევს state-of-the-art შედეგებს NoCaps-სა და COCO-ზე, ხოლო image-text retrieval-ში ITC/ITM/ITG მიზნებით fine-tuning იძლევა ძლიერ recall მაჩვენებლებს. BLIP-2 არ ავლენს ძლიერ in-context learning შესაძლებლობას, რადგან მისი წინასწარი სწავლების მონაცემებში თითო სემპლზე მხოლოდ ერთი image-text წყვილია, და დამატებით იგი მემკვიდრეობით იღებს გაყინული LLM-ების მიკერძოებებსა და ცოდნით შეზღუდვებს. საბოლოოდ, BLIP-2 შემოაქვს მოდულური და გამოთვლითად ეფექტიანი სტრატეგია მულტიმოდალური წინასწარი სწავლისთვის: გაყინული image encoder-ებისა და გაყინული LLM-ების შერწყმით და ორსაფეხურიანად გაწვრთნილი Querying Transformer-ის გამოყენებით, ჩარჩო აღწევს state-of-the-art შედეგებს მრავალ ამოცანაზე და მნიშვნელოვნად ამცირებს trainable პარამეტრების რაოდენობას.

2.3 ByT5

ByT5 არის Google-ის მიერ შექმნილი მოდელი, რომელიც ტექსტს ბაიტ დონეზე ამუშავებს, ტოკენიზატორის გარეშე. ეს მიდგომა განსაკუთრებით ეფექტურია ისეთ ენებში, სადაც ტოკენიზაცია რთულია (როგორიცაა ქართული).

ByT5-ის უპირატესობა:

1. შეუძლია ნებისმიერი ენის დამუშავება Unicode-ის ფარგლებში.
2. არ საჭიროებს წინასწარ განსაზღვრულ სიტყვების სიას.
3. კარგად მუშაობს მცირე ენებზე, სადაც არ არსებობს მაღალი ხარისხის ტოკენიზატორები.

თანამედროვე მსხვილი ენობრივი მოდელების უმრავლესობა მუშაობს ტოკენიზებულ ტექსტურ წარმოდგენებზე, სადაც ნედლი ტექსტი მოდელში მიწოდებამდე იყოფა სიტყვებად ან ქვესიტყვოვან ერთეულებად. მიუხედავად იმისა, რომ ქვესიტყვოვანი ტოკენიზაციის სტრატეგიები (მაგალითად, Byte Pair Encoding – (Sennrich,Haddow,Birch/სენრიხი,ჰადოუ,ბირჩი. 2016); WordPiece – (Wu,Schuster,Chen,Le,Norouzi,Macherey/უ,შუსტერი,ჩენ,ლე,ნორუზი,მახერეი. 2016); SentencePiece – (Kudo,Richardson/კუდო,რიჩარდსონი. 2018)) ამცირებს ფიქსირებული სიტყვების ლექსიკონთან დაკავშირებულ კლასიკურ out-of-vocabulary (OOV) პრობლემას, ისინი მაინც საჭიროებენ დამატებით წინასწარ დამუშავებას და მგრძნობიარენი არიან

ორთოგრაფიული ვარიაციების, ორთოგრაფიული შეცდომებისა და მრავალენოვანი განზოგადების სირთულეების მიმართ. ტრადიციულად, სიტყვაზე დაფუძნებული მოდელები ეყრდნობოდა წინასწარ განსაზღვრულ ლექსიკონს. თუ ტექსტში ჩნდებოდა ახალი ან იშვიათი სიტყვა, რომელიც ლექსიკონში არ იყო, მოდელი ვერ ამუშავებდა მას სწორად. ქვესიტყვოვანმა მიდგომებმა ეს პრობლემა შეამსუბუქა, რადგან სიტყვები დაიშალა უფრო მცირე ერთეულებად (მაგალითად, მორფემების ან სიმბოლოების კომბინაციებად), რაც საშუალებას იძლევა იშვიათი სიტყვების წარმოდგენა უკვე არსებული ერთეულების კომბინაციით. თუმცა, ამ პროცესს სჭირდება ლექსიკონის აგება დიდი კორპუსის საფუძველზე, სეგმენტაციის ალგორითმების კონფიგურაცია და ხშირად — ენის სპეციფიკური პარამეტრების განსაზღვრა. ByT5-ის ჩარჩო გვთავაზობს ფუნდამენტურად განსხვავებულ მიდგომას: ტოკენიზაციის სრულად ამოღებას და ტექსტის პირდაპირ UTF-8 ბაიტურ მიმდევრობებად წარმოდგენას. UTF-8 კოდირება წარმოადგენს უნივერსალურ სტანდარტს, რომელიც ყველა თანამედროვე ენის სიმბოლოს ასახავს ბაიტების სერიად. ByT5 ტექსტს აღიქვამს არა სიტყვებად ან ქვესიტყვებად, არამედ უშუალოდ ბაიტების მიმდევრობად. ამგვარი წარმოდგენა გამორიცხავს ლექსიკონის კონსტრუქციის საჭიროებას, სეგმენტაციის ჰეურისტიკებსა და ენის სპეციფიკურ წინასწარ დამუშავებას. ამ მიდგომას რამდენიმე მნიშვნელოვანი უპირატესობა აქვს:

1. უნივერსალური ენობრივი დაფარვა — რადგან UTF-8 ყველა ენას მოიცავს, მოდელს არ სჭირდება ცალკე ადაპტაცია ახალი ენისთვის.

2. ორთოგრაფიული გამძლეობა — ბაიტურ დონეზე მუშაობა ამცირებს მგრძნობიარობას წერის შეცდომების, არასტანდარტული ფორმების ან დიალექტური ვარიაციების მიმართ.
3. არქიტექტურული გამარტივება — სისტემიდან იშლება ტოკენიზაციის ცალკე კომპონენტი, რაც ამარტივებს მთლიანი მილსადენის დიზაინს.

თუმცა ბაიტურ დონეზე მოდელირებას მნიშვნელოვანი გამოთვლითი გამოწვევა ახლავს. ბაიტური მიმდევრობები, როგორც წესი, ბევრად გრძელია, ვიდრე ქვესიტყვოვანი ტოკენების მიმდევრობები. მაგალითად, ერთი სიმბოლო ინგლისურ ენაში ჩვეულებრივ შეესაბამება ერთ ბაიტს, მაგრამ მრავალი ენის (მაგალითად, ქართული, ჩინური ან არაბული) სიმბოლო UTF-8-ში შეიძლება წარმოდგენილი იყოს ორ ან მეტ ბაიტად. შედეგად, ტექსტის სიგრძე მნიშვნელოვნად იზრდება. Transformer-ზე დაფუძნებულ მოდელებში

(Vaswani,Shazeer,Parmar,Uszkoreit,Jones,Gomez/ვასვანი,შაზირი,პარმარი,უსკორეიტი,ჯოუნ

სი,გომესი. 2017) ყურადღების (self-attention) მექანიზმი კვადრატული სირთულისაა მიმდევრობის სიგრძის მიმართ ($O(n^2)$). ეს ნიშნავს, რომ მიმდევრობის სიგრძის ზრდასთან ერთად გამოთვლითი ღირებულება და მეხსიერების მოთხოვნა სწრაფად მატულობს. შესაბამისად, ბაიტურ დონეზე მუშაობა თეორიულად ზრდის რესურსულ მოთხოვნებს. ByT5-ის მთავარი წვლილი იმაში მდგომარეობს, რომ ავტორებმა აჩვენეს — თითქმის უცვლელი Transformer encoder-decoder არქიტექტურა ეფექტიანად ამუშავებს ბაიტურ შემომავალ მონაცემებს და ინარჩუნებს კონკურენტულ შესრულებას სტანდარტულ NLP ამოცანებზე. ეს შედეგი მნიშვნელოვანია, რადგან ადასტურებს, რომ რთული ტოკენიზაციის სქემები ყველა შემთხვევაში აუცილებელი არ არის და რომ ზოგადი არქიტექტურული მიდგომა შეიძლება საკმარისად ძლიერი იყოს უფრო ფუნდამენტურ, ბაიტურ წარმოდგენებთან სამუშაოდ. ამრიგად, ByT5 წარმოადგენს კონცეპტუალურ გადახვევას ტრადიციული ტოკენიზაციაზე დაფუძნებული NLP პრაქტიკიდან და გვთავაზობს უფრო უნივერსალურ, გამარტივებულ და ენობრივად ნეიტრალურ ჩარჩოს ტექსტის მოდელირებისთვის. სიმბოლოზე (character-level) დაფუძნებულ მოდელირებას ღრმა ისტორიული ფესვები აქვს ნეირონულ ენობრივ მოდელირებაში. ადრეული რეკურენტული ნეირონული ქსელები, როგორიცაა (Sutskever,Martens,Hinton/სუცკევერი,მარტენსი,ჰინტონი. 2011)-ისა და (Graves/გრეივსი. 2013)-ის ნამუშევრები, უშუალოდ სიმბოლოების მიმდევრობებზე მუშაობდნენ. იმ პერიოდში ფართოდ გამოიყენებოდა RNN და LSTM არქიტექტურები, რომლებიც ტექსტს ამუშავებდნენ სიმბოლო-სიმბოლოს დონეზე და ცდილობდნენ მომდევნო სიმბოლოს პროგნოზირებას. ასეთი მიდგომა ბუნებრივად გამორიცხავდა out-of-vocabulary პრობლემას, რადგან ლექსიკონი მხოლოდ სიმბოლოებისგან შედგებოდა. მოგვიანებით, მრავალი არქიტექტურული ინოვაცია პირველად სწორედ სიმბოლოს დონეზე ტესტირდებოდა, რადგან ეს ამარტივებდა ლექსიკონის მართვას და იძლეოდა მოდელის უნარის შეფასებას მორფოლოგიურად მდიდარ ან არასტანდარტულ ტექსტებზე. მიუხედავად ამისა, პრაქტიკაში სიტყვაზე და ქვესიტყვოვან ერთეულებზე დაფუძნებული ტოკენიზაცია გახდა დომინანტური სტანდარტი. ამის მთავარი მიზეზი იყო გამოთვლითი ეფექტიანობა: სიმბოლოს დონეზე მიმდევრობები ბევრად გრძელია, ხოლო გრძელი მიმდევრობები ზრდის როგორც მეხსიერების მოთხოვნას, ისე გამოთვლით ხარჯს. ქვესიტყვოვანი ტოკენიზაცია ამცირებს მიმდევრობის სიგრძეს და ამით ამარტივებს ტრენინგს დიდ მოდელებში. ჰიბრიდულმა „character-aware“ მიდგომებმა სცადეს ამ ორი სამყაროს შერწყმა. მაგალითად: • ELMo

(Peters,Neumann,Iyyer,Gardner,Clark,Lee/პიტერსი,ნოიმანი,აიერი,გარდნერი,კლარკი,ლი. 2018) — იყენებდა სიმბოლოებზე დაფუძნებულ კონვოლუციურ ნეირონულ ქსელს (CNN), რათა შეექმნა სიტყვების წარმოდგენები, რომლებიც შემდეგ გადაეცემოდა ბიდირექციულ LSTM-ს. ამ გზით მოდელი იღებდა მორფოლოგიურ ინფორმაციას სიმბოლოების დონიდან, თუმცა მაინც მუშაობდა სიტყვების საზღვრებში. • CharacterBERT (El Boukkouri,Gao,Mehdad,Ma,Reynolds/ელ ბუკური,გაო,მეჰდადი,მა,რეინოლდსი. 2020) — ანაცვლებდა WordPiece ემბედინგებს სიმბოლოზე დაფუძნებული ემბედინგებით, თუმცა ინარჩუნებდა BERT-ის ტოკენურ სტრუქტურას. შედეგად, მოდელი უკეთ უმკლავდებოდა ორთოგრაფიულ ვარიაციებსა და იშვიათ სიტყვებს, მაგრამ სრულად არ უარყოფდა ტოკენიზაციას. უფრო ახლახან, CANINE (Clark,Garrette,Turbax,Kumar,Saleh,Smith/კლარკი,გარეტი,ტარბაქსი,კუმარი,სალეჰი,სმიტი. 2021)-მა შეისწავლა ტოკენიზაციის გარეშე მოქმედი ენკოდერის არქიტექტურა. CANINE ტექსტს სიმბოლოების დონეზე ამუშავებდა და იყენებდა სპეციალურ ქვე-სემპლინგისა და ჰიერარქიული დამუშავების მექანიზმებს, რათა შეემცირებინა გრძელი მიმდევრობების გამოთვლითი ღირებულება. ეს მნიშვნელოვანი ნაბიჯი იყო სრულად ტოკენიზაციისგან დამოუკიდებელი მოდელებისკენ. ByT5 ამ მიმართულებას კიდევ უფრო ავითარებს. იგი ადაპტირებს სრულ T5 (Text-to-Text Transfer Transformer) ენკოდერ-დეკოდერ ჩარჩოს (Raffel,Shazeer,Roberts,Lee,Narang,Matena/რაფელი,შაზირი,რობერტსი,ლი,ნარანგი,მატენა. 2020; Xue,Constant,Roberts,Kale,Al-Rfou,Siddhant/სიუ,კონსტანტი,რობერტსი,კეილე,ალ-რფუ,სიდჰანტი. 2021) ისე, რომ მოდელი მთლიანად ბაიტურ დონეზე მუშაობდეს. განსხვავებით მხოლოდ ენკოდერზე ორიენტირებული სისტემებისგან, T5 წარმოადგენს უნივერსალურ ტექსტ-ტექსტ არქიტექტურას, რომელიც გამოიყენება თარგმნისთვის, შეჯამებისთვის, კითხვებზე პასუხის გაცემისთვის და სხვა ამოცანებისთვის ერთიანი ფორმატით. ByT5-ის მნიშვნელოვანი მიღწევა არის ის, რომ ეს მიდგომა წარმატებით სკალირდება ძალიან დიდ მოდელებამდე — 10 მილიარდზე მეტი პარამეტრით. ეს აჩვენებს, რომ ბაიტურ დონეზე მუშაობა არ არის შეზღუდული მცირე ან ექსპერიმენტული მოდელებით; პირიქით, იგი თავსებადია თანამედროვე მასშტაბური ტრანსფორმერული არქიტექტურების პარადიგმასთან. საერთო ჯამში, სიმბოლოებზე დაფუძნებული მოდელირების ისტორიული ტრადიცია გადაიქცა თანამედროვე, მასშტაბურ ბაიტურ მოდელებად. ByT5 წარმოადგენს ამ ევოლუციის ერთ-ერთ მნიშვნელოვან ეტაპს — აერთიანებს ტოკენიზაციის გარეშე მუშაობის იდეას და სრულმასშტაბიან encoder-decoder ტრანსფორმერულ არქიტექტურას, რაც ამტკიცებს, რომ

ენობრივი მოდელირება შეიძლება განხორციელდეს უფრო ფუნდამენტურ, უნივერსალურ ტექსტურ წარმოდგენებზე დაყრდნობით. ByT5 წარმოიშვა მრავალენოვანი T5 (mT5) მოდელიდან მინიმალური სტრუქტურული ცვლილებებით და ინარჩუნებს მის ძირითად encoder-decoder ტრანსფორმერულ არქიტექტურას (Raffel,Shazeer,Roberts,Lee,Narang,Matena/რაფელი,შაზირი,რობერტსი,ლი,ნარანგი,მატენა. 2020; Xue,Constant,Roberts,Kale,Al-Rfou,Siddhant/სიუ,კონსტანტი,რობერტსი,კეილე,ალ-რფუ,სიდჰანტი. 2021). მთავარი განსხვავება არის ის, რომ SentencePiece ტოკენიზატორი სრულად ამოღებულია და მის ნაცვლად გამოიყენება პირდაპირი UTF-8 ბაიტური მიმდევრობები, რის შედეგადაც ლექსიკონი შედგება მხოლოდ 256 შესაძლო ბაიტური მნიშვნელობისგან (0–255) და დამატებით სამი სპეციალური იდენტიფიკატორისგან: padding, end-of-sequence და <UNK>. წინასწარი სწავლების ამოცანა უცვლელია და ეფუძნება span corruption მიდგომას, რომელიც პირველად T5-ში დაინერგა (Raffel,Shazeer,Roberts,Lee,Narang,Matena/რაფელი,შაზირი,რობერტსი,ლი,ნარანგი,მატენა. 2020), თუმცა განსხვავება ისაა, რომ masking ხორციელდება ბაიტურ სპანებზე და არა ქვესიტყვოვან ტოკენებზე; თუ mT5 საშუალოდ ფარავს სამ ტოკენს, ByT5 ზრდის საშუალო სპანის სიგრძეს 20 ბაიტამდე, რაც ამცირებს ტრივიალური რეკონსტრუქციის შესაძლებლობას და აიძულებს მოდელს გრძელი კონტექსტური დამოკიდებულებების სწავლას. არქიტექტურულად მნიშვნელოვანი ცვლილებაა encoder-ისა და decoder-ის სიღრმის გამიჯვნა 3:1 თანაფარდობით, რის შედეგადაც ენკოდერი ხდება შედარებით უფრო მძიმე; ეს გადაწყობა კომპენსაციას უწევს იმ ფაქტს, რომ ბაიტურ ლექსიკონს არ სჭირდება დიდი embedding მატრიცა და პარამეტრები გადანაწილდება ტრანსფორმერის შიდა ფენებში, რაც საშუალებას აძლევს მოდელს ლექსიკური წარმოდგენები ისწავლოს dense ფენებში. ტოკენზე დაფუძნებულ მოდელებში პარამეტრების მნიშვნელოვანი ნაწილი განთავსებულია embedding და output softmax მატრიცებში; მაგალითად, mT5-Base-ში დაახლოებით ორი მესამედი პარამეტრებისა ლექსიკონთან არის დაკავშირებული (Xue,Constant,Roberts,Kale,Al-Rfou,Siddhant/სიუ,კონსტანტი,რობერტსი,კეილე,ალ-რფუ,სიდჰანტი. 2021), მაშინ როცა ByT5 ამ რესურსს ტრანსფორმერულ ფენებში ანაწილებს და ინარჩუნებს შესაბამის საერთო პარამეტრულ ზომებს სხვადასხვა სკალაზე. ბაიტური მიმდევრობები გრძელია ქვესიტყვოვანთან შედარებით, რის გამოც Transformer-ის ყურადღების კვადრატული სირთულე ზრდის გამოთვლით ხარჯს; ემპირიულად, ByT5 მოითხოვს დაახლოებით 1.2-ჯერ მეტ floating-point ოპერაციას წინასწარი სწავლებისას და აღწევს დაახლოებით 75%-იან throughput-ს mT5-თან შედარებით იმავე ჰარდ웨어ზე,

ხოლო ინფერენსის შენელება ამოცანაზეა დამოკიდებული და განსაკუთრებით შესამჩნევია გრძელი მიმდევრობების კლასიფიკაციაში. GLUE და SuperGLUE ბენჩმარკებზე (Wang,Singh,Michael,Hill,Levy,Bowman/ვანგი,სინგჰი,მაიქლი,ჰილი,ლევი,ბოუმანი. 2018; Wang,Pruksachatkun,Nangia,Singh,Michaels,He/ვანგი,პრუქსაჩატკუნი,ნანგია,სინგჰი,მაიქლსი,ჰე. 2019) ByT5 აჭარბებს mT5-ს მცირე და საშუალო ზომის მოდელებში, თუმცა ოდნავ ჩამორჩება დიდ სკალებზე, რაც აიხსნება პარამეტრების უფრო ეფექტური გამოყენებით მცირე მოდელებში, რადგან dense ფენები ცვლიან იშვიათად გამოყენებად embedding მატრიცებს. GEM-XSum (Gehrmann,Adewumi,Aggarwal,Ammanamanchi,Anuoluwapo,Bosselut/გერმანი,ადევუმი,აგგარვალი,ამმანამანჩი,ანუოლუვაპო,ბოსელუტი. 2021), TweetQA (Xiong,Wu,Wang,Kulkarni,Yu,Chang/სიონგი,უ,ვანგი,კულკარნი,იუ,ჩანგი. 2019) და DROP (Dua,Wang,Dasigi,Stanovsky,Singh,Gardner/დუა,ვანგი,დასიგი,სტანოვსკი,სინგჰი,გარდნერი. 2019) ამოცანებზე ByT5 სისტემატურად აჭარბებს mT5-ს ყველა ზომის მოდელში, რაც მიუთითებს, რომ ბაიტურ დონეზე მუშაობა არ ამცირებს გენერაციულ ფლუენტურობას და შესაძლოა აუმჯობესებდეს არაოფიციალური ან ხმაურიანი ტექსტის დამუშავებას. XTREME ამოცანებზე (Hu,Ruder,Siddhant,Neubig,Firat,Johnson/ჰუ,რუდერი,სიდჰანტი,ნიუბიგი,ფირატი,ჯონსონი. 2020) ByT5 კონკურენტულია mT5-თან და ზოგ შემთხვევაში აჭარბებს მას ერთენოვან მრავალამოცანიან კონფიგურაციებში, ხოლო ენებს შორის განსხვავებები სუსტად კორელირებს SentencePiece-ის შეკუმშვის მაჩვენებელთან, რაც მიუთითებს, რომ ტოკენიზაციის გრანულარობა გავლენას ახდენს ენობრივ შედეგებზე. ტრანსლიტერაციის (Roark,Wolf,Do,Bhattacharya,Ghosh,Kirov/როარკი,ვულფი,დუ,ბჰატაჩარია,გოშ,კიროვი. 2020), grapheme-to-phoneme კონვერსიის (Gorman,Ashby,Zhu,Belinkov,Cotterell,Hulden/გორმანი,ეშბი,ჟუ,ბელინკოვი,კოტერელი,ჰულდენი. 2020) და მორფოლოგიური ინფლექციის (Vylomova,White,Salesky,Mielke,Wu,Ponti/ვილომოვა,უაიტი,სეილზკი,მიელკე,უ,პონტი. 2020) ამოცანებზე ByT5 აღწევს მნიშვნელოვან გაუმჯობესებებს, რადგან ბაიტურ მიმდევრობებზე პირდაპირი წვდომა აუმჯობესებს ორთოგრაფიულად და ფონოლოგიურად სტრუქტურირებული ამოცანების მოდელირებას. ხელოვნური კორუფციის ექსპერიმენტებში, რომლებიც მოიცავდა სიმბოლოს წაშლას, გამეორებას, uppercase ტრანსფორმაციას, შემთხვევით რეგისტრს და antspeak ფორმატირებას, ByT5 მნიშვნელოვნად ნაკლებ დეგრადაციას აჩვენებს mT5-თან შედარებით, რადგან ტოკენის

მკაცრი საზღვრების არარსებობა ზრდის მოქნილობას პერტურბაციების მიმართ. აბლაციის ექსპერიმენტები ადასტურებს, რომ მძიმე ენკოდერის კონფიგურაცია აუმჯობესებს შედეგებს, დაბალანსებული encoder-decoder არქიტექტურები ჩამორჩება ანალოგიური პარამეტრული ზომის შემთხვევაშიც, გრძელი masked სპანები სასარგებლოა გენერაციული ამოცანებისთვის, ხოლო მოკლე სპანები ოდნავ უკეთესია კლასიფიკაციაში; სიმბოლოზე დაფუძნებული ლექსიკონები კონკურენტულია, თუმცა ჩამორჩება ბაიტურ მოდელირებას, სავარაუდოდ პარამეტრების განაწილების ნაკლებად ეფექტურობისა და UTF-8-ის ერთგვაროვანი სივრცის უპირატესობის გამო. წინასწარი სწავლების დრო იზრდება დაახლოებით 33%-ით mT5-თან შედარებით, ხოლო ინფერენსის შენელება ამოცანაზეა დამოკიდებული და ზოგიერთ კლასიფიკაციურ სცენარში შესაძლოა მნიშვნელოვნად გაიზარდოს, თუმცა გაუმჯობესებული გამძლეობა და მრავალენოვანი განზოგადება შესაძლოა ამ გამოთვლით კომპრომისს ამართლებდეს; საბოლოოდ, ByT5 აჩვენებს, რომ მასშტაბური, ტოკენიზაციის გარეშე ენობრივი მოდელირება პრაქტიკულად განხორციელებადი და კონკურენტულია, ხოლო მომავალი მიმართულებები, როგორიცაა sparse attention, ქვე-სემპლინგი და უფრო ეფექტიანი არქიტექტურები, პოტენციურად შეამცირებს ლატენტურობას და გააუმჯობესებს ეფექტიანობას.

2.4 შეზღუდვები ქართულ NLP-ში

ქართული ენისთვის არსებული ძირითადი პრობლემებია:

1. მცირე მოცულობის მონაცემები,
2. კორპუსების ფრაგმენტულობა,
3. აკადემიური რესურსების ნაკლებობა,
4. საერთაშორისო მოდელებში ქართული მხოლოდ მინიმალურადაა წარმოდგენილი.

2.5 მონაცემთა შეგროვება, როგორც მეთოდოლოგიური ნაწილი

პროექტის განხორციელება მოითხოვს მინიმუმ 300,000 სურათისა და შესაბამისი ქართული აღწერის მონაცემთა ბაზას. მონაცემები მოიცავს:

1. ყოველდღიური სცენები,

2. საგანმანათლებლო ვიზუალები,
3. კულტურულ-სიმბოლური სურათები,
4. ტურისტული ობიექტები.

2.6 მონაცემთა წინამუშავება

1. სურათების ზომის სტანდარტიზაცია: 224×224.
2. ფერის კორექცია და noise reduction.
3. ტექსტების ორთოგრაფიული ნორმალიზაცია.
4. Train/Validation/Test (80/10/10) გაყოფა.

2.7 მოდელების სინქრონიზაცია

1. BLIP2 encoder იღებს სურათს და აგენერირებს ვიზუალურ ტენსორს.
2. Projection Layer ადაპტირებს ვექტორის ზომას ByT5-ის შესასვლელთან.
3. ByT5 decoder ავტორეგრესიულად ქმნის ტექსტს ქართულ ენაზე.

2.8 Martha მოდელის მათემატიკური მოდელირება

პროექტის „Martha“-ს მთავარი ინოვაციაა ორი სხვადასხვა არქიტექტურის სინთეზი: BLIP2-ის Encoder და ByT5-ის Decoder. მათემატიკური მოდელირება გვაძლევს საშუალებას ფორმალურად დავწეროთ პროცესები, რომლითაც სურათი გადაიქცევა ქართულ ენაზე ტექსტურ აღწერად.

ზოგადი მოდელირების სქემა

Encoder–Decoder არქიტექტურის ზოგადი ჩანაწერი:

$$z=f\theta(x), y=g\phi(z)$$

სადაც:

$x \in R^{H \times W \times 3}$ არის შეყვანილი RGB სურათი,

fθ Encoder (BLIP2), პარამეტრებით θ\thetaθ,

$z \in R^d$– სემანტიკური ვექტორი (ვიზუალური ტენსორი),

gφ – Decoder (ByT5), პარამეტრებით φ\phiφ,

y=(y1,y2,…,yT) – გამომავალი ტექსტი ქართულ ენაზე.

2.9 Encoder – BLIP2

BLIP2 იყენებს Vision Transformer (ViT)-ს და Q-Former-ს. სურათი x წინასწარ იყოფა პატჩებად:

$$x \rightarrow \{p1, p2, \ldots, pN\}, p_i \in R^{P \times P \times 3}$$

ViT გარდაქმნის თითოეულ პატჩს ვექტორად:

$$h_i = E_{ViT}(p_i), h_i \in R^d$$

Q-Former იყენებს Query Token-ებს (qjq_jqj), რომლებიც სემანტიკურად აჯამებენ ვიზუალურ ინფორმაციას:

$$z_j = \text{Attention}(q_j, \{h1, \ldots, h_N\})$$

საბოლოოდ, Encoder აბრუნებს ვიზუალურ ტენსორს:

$$Z = (z1, z2, \ldots, zM), Z \in R^{M \times d}$$

2.10 პროექციის შრე (Projection Layer)

რადგან ByT5-ის შეყვანილი სივრცე (d′) განსხვავდება BLIP2-ის გამოსავლისგან (d), საჭიროა პროექცია:

$$h_j = W_{zj} + b, \quad W \in R^{d' \times d}, b \in R^{d'}$$

ამით ვიღებთ ახალ ვექტორულ წარმოდგენას:

$$H=(h1,h2,\ldots,hM), \qquad H \in R^{M \times d'}$$

2.11 Decoder – ByT5

ByT5 არის Autoregressive Transformer Decoder, რომელიც ტექსტს ბაიტ დონეზე ამუშავებს. ტექსტის გენერაციის პროცესი მოდელირდება შემდეგნაირად:

$$P(y_t|y<t,H)=\text{softmax}(W_o s_t))$$

სადაც:

y_t – t-წამყვანი სიმბოლო (byte-level token),

s_t – decoder hidden state,

H – Encoder-ის პროექცირებული გამოსავალი,

Wo – output matrix.

ტექსტის გენერაცია ხდება ავტორეგრესიულად:

$$y_t=\text{argmax}P(y_t|y<t,H)$$

2.12 დანაკარგის ფუნქცია (Loss Function)

მიზანი არის სწორი ტექსტის მაქსიმალური ალბათობის მინიჭება მოცემული სურათისთვის:

$$L(\theta,\phi,W,b)=-\sum_{t=1}^{T} \log P(yt|y<t,H)$$

ეს არის Cross-Entropy Loss, სადაც თითოეული სიმბოლოს პროგნოზი შეადარება ნამდვილ სიმბოლოს.

Padding token-ები იგნორირებულია (–100-100–100 ჩანაცვლებით).

Label smoothing (ϵ) შეიძლება გამოყენებულ იქნას:

$$L_{smooth}=(1-\epsilon)L_{CE}+\epsilon\cdot U$$

სადაც U – ერთგვაროვანი განაწილებაა.

2.12 რეგულარიზაცია

პროექტში გამოიყენება რამდენიმე რეგულარიზაციის ტექნიკა:

1. Dropout – შემთხვევით აჩერებს ნეირონებს:

$$h'=(m\odot h)/p \text{ , } m\sim Bernoulli(p)$$

2. Weight Decay – პარამეტრების ნორმის დაჯარიმება:

$$L_{final}=L+\lambda\|\theta\|^2_2$$

3. Gradient Clipping – თავიდან იცილებს Gradient Explosion-ს:

$$g'=g/\max(1,\|g\|/\tau)$$

2.13 ოპტიმიზატორი

გამოიყენება AdamW ოპტიმიზატორი:

$$m_t=\beta_1 m_{t-1}+(1-\beta_1)g_t$$

$$v_t=\beta_2 v_{t-1}+(1-\beta_2)g_t^2$$

$$\theta_{t+1}=\theta_t-\eta\ (m_t/(\sqrt{\mathbf{vt}}+\epsilon))-\lambda\eta\theta_t$$

სადაც λ – Weight Decay კოეფიციენტია.

2.14 სწავლების სიხშირის დამგეგმავი (Learning Rate Scheduler)

გამოიყენება ReduceLROnPlateau:

> თუ Validation Loss არ უმჯობესდება N ეპოქის განმავლობაში, Learning Rate ნახევრდება.

ფორმალურად:

$$\eta_{t+1}=\{\ \eta_t\cdot\gamma, \text{ თუ no improvement for } N \mid \eta_t, \text{ სხვა შემთხვევაში}$$

2.15 ინტერპრეტაცია

ამ მოდელირების შედეგად:

1. Encoder სწავლობს ვიზუალური სამყაროს კომპაქტურ წარმოდგენას.
2. Projection Layer უზრუნველყოფს ენობრივ და ვიზუალურ სივრცეებს შორის თავსებადობას.
3. Decoder ქმნის ქართულ ტექსტს, რომელიც ენათმეცნიერებრივად სწორი და კონტექსტურად შესაბამისია.(Li,Li,Suvarese,Hoi/ლი,ლი სუვარესი,ჰოი. 2023) (Xu,Ba,Kiros,Cho,Courville,Salakhutdinov,Zemel,Bengio/ქსუ,ბა,კიროსი,ჩო,კორვილი,სალახუტდინოვი,ზემელ,ბენგიო. 2015)

თავი III. კვლევის ექსპერიმენტული ნაწილი

პროექტის „Martha“-ს წარმატებისთვის კრიტიკულად მნიშვნელოვანია ექსპერიმენტული ნაწილის დეტალურად დაგეგმვა და განხორციელება. ეს სექცია აღწერს მონაცემთა სტრუქტურას, გაყოფის სტრატეგიას, გამოყენებულ ჰიპერპარამეტრებსა და მონაცემთა გაძლიერების (augmentation) შედეგებს.

3.1 მონაცემთა ბაზა

პროექტისათვის შექმნილია 300,000 სურათისა და ქართული კაპშენის მონაცემთა ბაზა, რომელიც მოიცავს სხვადასხვა კატეგორიას:

1. ყოველდღიური ცხოვრების სცენები,
2. საგანმანათლებლო და სამეცნიერო გამოსახულებები,
3. კულტურული და ისტორიული ობიექტები,
4. ტურისტული და ბუნებრივი პეიზაჟები,
5. ადამიანთა სოციალური ინტერაქციები.

ამ მრავალფეროვანი მონაცემების გამოყენება უზრუნველყოფს მოდელის გენერალიზაციას და მისი შედეგების გამოყენებას რეალურ სცენარებში.

3.2 მონაცემთა გაყოფა

მონაცემები იყოფა შემდეგნაირად:

1. Train Set – 80% (~240,000 სურათი) გამოიყენება მოდელის პარამეტრების დასასწავლად.
2. Validation Set – 10% (~30,000 სურათი) გამოიყენება მოდელის პარამეტრების შერჩევისა და overfitting-ის კონტროლისთვის.
3. Test Set – 10% (~30,000 სურათი) გამოიყენება საბოლოო შეფასებისთვის, unseen მონაცემებზე.

ეს სტრატეგია უზრუნველყოფს როგორც საკმარის სასწავლო მასალას, ასევე სანდო შეფასებას.

3.3 მონაცემთა წინამუშავება

1. სურათების რეზოლუცია: ყველა სურათი გადაყვანილია 224×224 ფორმატში, რათა თავსებადი იყოს Encoder-ის არქიტექტურასთან.
2. ფერის კორექცია: გამოიყენება normalize ოპერაციები, რათა სურათები ერთიან დიაპაზონში მოვიდეს.
3. ტექსტების დამუშავება: ყველა კაპშენი გაიარა ორთოგრაფიული და გრამატიკული ნორმალიზაცია.

3.4 აუგმენტაცია

Train მონაცემებზე გამოყენებულია მონაცემთა გაძლიერების ტექნიკები, რათა მოდელმა შეძლოს რეალური გარემოს მრავალფეროვნების სწავლის უნარი:

1. RandomResizedCrop (0.8–1.0 მასშტაბი),
2. Horizontal Flip (50% ალბათობით),
3. Color Jitter (სინათლე, კონტრასტი, ტონი).

Validation და Test სეტებზე გამოყენებულია მხოლოდ Resize და Center Crop, რათა შეფასება იყოს სტაბილური და ობიექტური.

3.5 ჰიპერპარამეტრები

ექსპერიმენტული ტრენინგი დაფუძნებულია შემდეგ კონფიგურაციაზე:

Batch Size: 16

Epochs: 100 (Early Stopping: 10 ეპოქის გაუმჯობესების გარეშე)

Optimizer: AdamW

Learning Rate:

1. Projection Layer, Q-Former → 1×10^{-5}
2. Decoder (ByT5) → 5×10^{-5}

Weight Decay: 0.01

Scheduler: ReduceLROnPlateau (patience = 2, factor = 0.5)

Gradient Clipping: 1.0

ეს პარამეტრები შერჩეულია ექსპერიმენტულად, რათა ბალანსი მოიძებნოს სწავლის სიჩქარესა და მოდელის სტაბილურობას შორის.

3.6 ექსპერიმენტული შედეგების დაგეგმვა

პროექტის მიმდინარეობისას დაგეგმილია რამდენიმე ექსპერიმენტის ჩატარება:

1. Baseline მოდელი: მხოლოდ BLIP2 + ByT5 მინიმალური რეგულარიზაციით.
2. Augmented მოდელი: Augmentation ჩართულია.
3. Different Learning Rate Model: სხვადასხვა Learning Rate-ის ექსპერიმენტი (Grid Search).
4. Regularization Model: Dropout-ისა და Weight Decay-ის სხვადასხვა მნიშვნელობების ტესტირება.

ამ ექსპერიმენტების შედარება საშუალებას მოგვცემს დავადგინოთ, რომელი კონფიგურაციაა ყველაზე ეფექტური.

3.7 მოსალოდნელი შედეგები

მოსალოდნელია, რომ:

Augmentation-ით გაწვრთნილი მოდელი აჯობებს Baseline-ს.

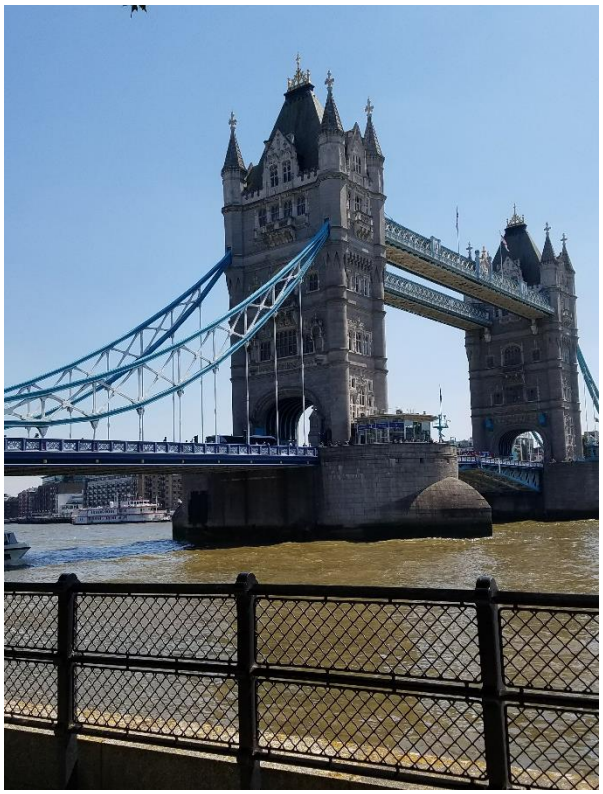
Regularization ტექნიკები შეამცირებს overfitting-ს.

მოდელი მიაღწევს მაღალი სიზუსტის მეტრიკებს (BLEU, METEOR, CIDEr).

თავი IV. შედეგები

წვრთვნა მიმდინარეობდა 50 ეპოქის განმავლობაში , წინასწარი გაჩერების კრიტერიუმით: 10 ნაბიჯით. წვრთვნა შეჩერდა მე23 ეპოქაზე საშუალო ვალიდაციისდანაკარგის მაჩვენებლით ~14% .

```
Fetching 2 files: 100% 2/2 [00:34<00:00, 17.16s/it]
Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.48it/s]
pytorch_model.bin: 100% 2.33G/2.33G [00:07<00:00, 292MB/s]
model.safetensors: 100% 2.33G/2.33G [00:03<00:00, 634MB/s]
generation_config.json: 100% 147/147 [00:00<00:00, 1.35MB/s]
Generated Caption: ლურჯი ჩიტი ზის ტოტზე თეთრი ყვავილებით
```

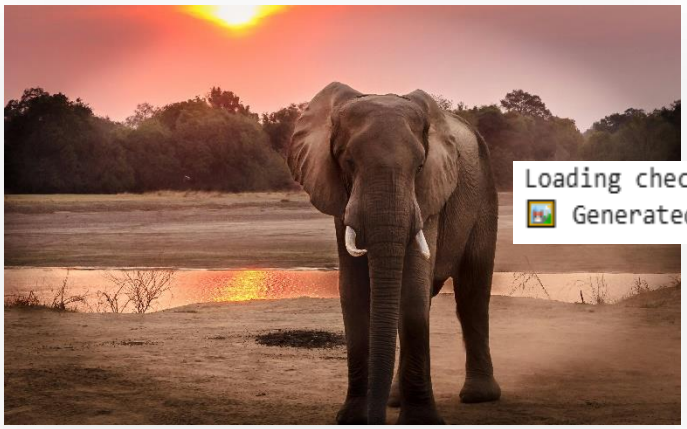


Generated Caption: თაუერ ბრიჯი ლონდონში, ინგლისი

Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.18it/s]
Generated Caption: საჭმლის თეფში მაგიდაზე

Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.73it/s]
Generated Caption: კურდღელი ბალახზე ზის შენობის წინ

Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.61it/s]
Generated Caption: ორი ოქროს რეტრივერი ბალახზე ზის



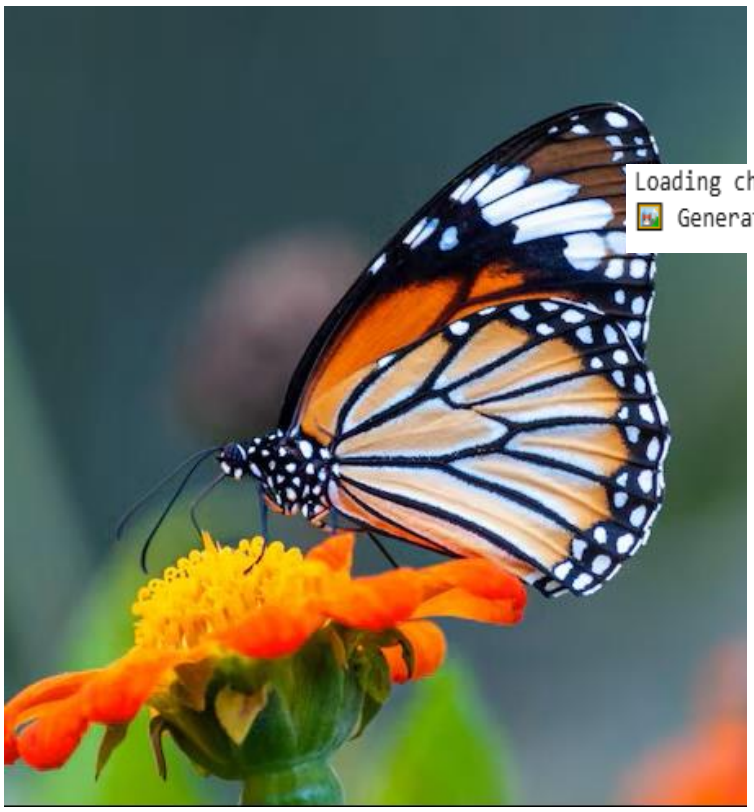
```
Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.13it/s]
Generated Caption: სპილო დგას მინდორში მზის ჩასვლისას
```

```
Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.35it/s]
Generated Caption: კაცი და ქალი ბავშვთან ერთად პოზირებენ ფოტოსთვის
```

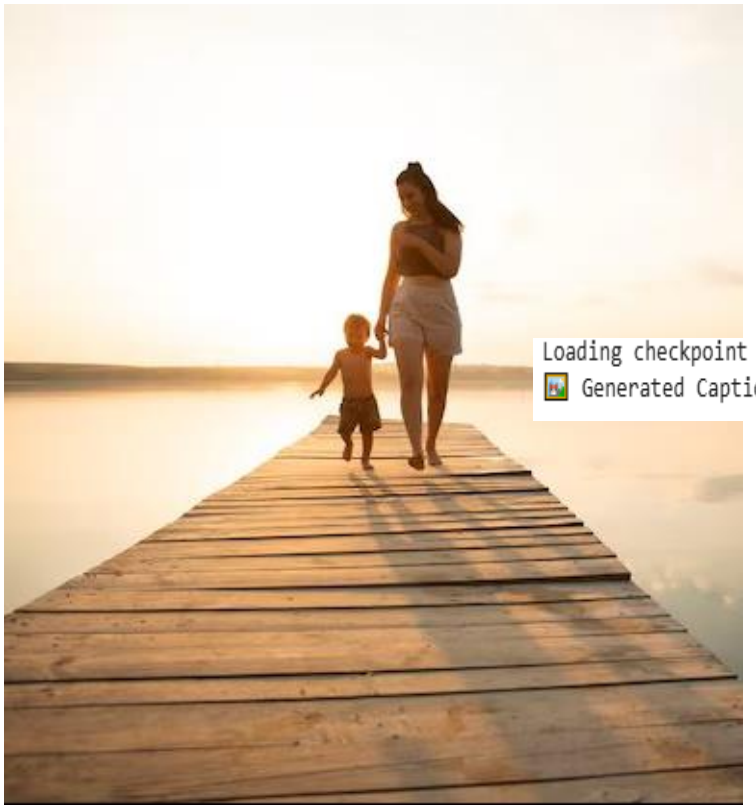
Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.68it/s]
Generated Caption: საბრძოლო თვითმფრინავი გაჩერებულია მიწდორში

Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.28it/s]
Generated Caption: წითელი სახანძრო მანქანა გაჩერებულია შენობის წინ

Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.24it/s]
Generated Caption: ქალი იღებს სურათს კამერით



Loading checkpoint shards: 100% 2/2 [00:00<00:00, 12.50it/s]
Generated Caption: მოწარქი პეპელა ვარდისფერ ყვავილზე



გადაწვრთვნის სკრიპტის დეტალური აღწერა

ფაილის დასაწყისი და იმპორტები

%%writefile train.py

import os, re, sys, csv, torch, random, numpy as np, pandas as pd

from torch.utils.data import Dataset, DataLoader, Sampler

from transformers import AutoTokenizer

from captioner import Martha

from tqdm import tqdm

ეს ნაწილი:

ქმნის Python სკრიპტს სახელად train.py;

იტვირთება ყველა საჭირო ბიბლიოთეკა:

1. torch, numpy, pandas — ძირითადი მონაცემების დამუშავება და ნერვული ქსელის ტრეინინგი;

2. Dataset, DataLoader — PyTorch-ის სტანდარტული dataset-მექანიზმი;
3. transformers.AutoTokenizer — ტექსტის ტოკენიზაცია (byt5 მოდელისთვის);
4. Martha მოდელი — მორგებული სურათის აღწერის მოდელი, აღწერილია captioner.py-ში;
5. tqdm — პროგრესბარის ჩვენება.

პრომპტების განსაზღვრა

prompt_variants = [

"რა ხდება სურათზე?", "რა ჩანს სურათში?", ...

]

ეს არის წინასწარ განსაზღვრული ქართული კითხვები/ინსტრუქციები, რომლითაც მოდელი სწავლობს სხვადასხვა ფორმით სურათის აღწერას. მიზანი: მოდელმა ისწავლოს მრავალფეროვანი ტონით და ფორმით პასუხის გაცემა, ანუ არა ერთი და იგივე სტრუქტურით.

მოწყობილობის არჩევა

device = "cuda" if torch.cuda.is_available() else "cpu"

ამოწმებს, არის თუ არა GPU ხელმისაწვდომი (CUDA), რათა ტრენინგი უფრო სწრაფად შესრულდეს.

Dataset-ის აღწერა

class ImageCaptionDataset(Dataset):

ეს კლასი განსაზღვრავს, როგორ ჩაიტვირთოს თითოეული ნიმუში (სურათი + აღწერა).

კონსტრუქტორი

def __init__(self, csv_path, tokenizer, split="train", prompt_variants=prompt_variants):

1. კითხულობს CSV ფაილს (georgian_captions_local_split.csv);
2. ფილტრავს მხოლოდ იმ სტრიქონებს, სადაც "split" ველია = "train" ან "val";

3. ინახავს ტოკენაიზერსა და პრომპტების სიას.

getitem

row = self.data[idx]

tensor_path = row["tensor_path"]

image_tensor = torch.load(tensor_path)

caption = row["caption_ka"]

prompt = random.choice(self.prompt_variants)

1. ტენზორად ტვირთავს სურათის წინასწარ დამუშავებულ მონაცემს (tensor_path);
2. იღებს ქართულ აღწერას;
3. არჩევს ერთ შემთხვევით პრომპტს.

შემდეგ:

prompt_tok = tokenizer(prompt, ...)

caption_tok = tokenizer(caption, ...)

ტოკენიზაცია — ტექსტის გარდაქმნა რიცხვების სერიად, რომელიც მოდელს ესმის.

და ბოლოს:

labels[labels == tokenizer.pad_token_id] = -100

— სვარდება padding ტოკენები, რათა მოდელის დაკარგვის ფუნქციაში არ დაითვალოს (CrossEntropy არ ითვლის -100 მნიშვნელობებს).

დაბრუნებული შედეგი: {image, decoder_input_ids, decoder_attention_mask, labels}

გზების შექმნა და Tokenizer

tokenizer = AutoTokenizer.from_pretrained("google/byt5-base")

byt5 გამოიყენება, რადგან იგი ბაიტ-დონეზე მუშაობს და უკეთ უჭერს მხარს ქართულ ტექსტს.

მონაცემთა ჩატვირთვა

train_dataset = ImageCaptionDataset(csv_path, tokenizer, split="train")

val_dataset = ImageCaptionDataset(csv_path, tokenizer, split="val")

ორი dataset: სწავლებისთვის და ვალიდაციისთვის.

FixedOrderSampler

class FixedOrderSampler(Sampler[int]):

სპეციალური Sampler, რომელიც:

1. იღებს ინდექსების ჩამონათვალს;
2. იწყებს კონკრეტული offset-დან (თუ ტრენინგი შეწყდა შუაში);
3. არ აბრუნებს შემთხვევით შერჩევას — ანუ შედეგი განმეორებადია.

ეს აუცილებელია „resume from checkpoint“-ისას, რომ მოდელი ზუსტად იმავე სემპლიდან გააგრძელოს სწავლა.

მოდელი, ოპტიმიზატორი და scheduler

model = Martha().to(device)

optimizer = torch.optim.AdamW([...])

scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(...)

1. მოდელი იტვირთება GPU-ზე;
2. AdamW ოპტიმიზატორი გამოიყენება სხვადასხვა ნაწილებისთვის განსხვავებული learning rate-ებით (decoder-ს უფრო მაღალი აქვს);
3. Scheduler ამცირებს learning rate-ს, როცა validation loss აღარ უმჯობესდება.

დამხმარე ფუნქციები

cleanup_checkpoints

შლის ძველ checkpoint ფაილებს, ტოვებს მხოლოდ ბოლო რამდენიმე მათგანს.

latest_checkpoint

პოულობს ბოლო (უახლეს) checkpoint-ს.

make_epoch_indices

ქმნის deterministic (განსაზღვრულად შემთხვევით) ინდექსების სიას თითოეული ეპოქისთვის, რომ შედეგები გამეორებადი იყოს.

კონფიგურაცია და checkpoint-დან გაგრძელება

ეს ნაწილი ამოწმებს:

1. იყო თუ არა წინა გაწვრთნა;
2. რომელი epoch იყო ბოლო;
3. რამდენი batch დასრულდა.

თუ checkpoint არსებობს:

1. იტვირთება მოდელი და ოპტიმიზატორის მდგომარეობა;
2. აღდგება RNG (random, numpy, torch) seed-ები ზუსტად იგივე სიტუაციაში დასაბრუნებლად;
3. ადგენს, სად უნდა განაგრძოს ტრენინგი.

თუ არა — იწყებს თავიდან.

DataLoader-ების შექმნა

def make_train_loader(indices, resume_batch, batch_size):

ამზადებს DataLoader-ს ისე, რომ ზუსტად იმ batch-დან დაიწყოს, საიდანაც შეწყდა.

Validation loader კი უბრალოდ სრულად გადის ყველა ნიმუშზე.

მთავარი ტრეინინგ ციკლი

for epoch in range(start_epoch, epochs):

ძირითადი ტრეინინგ ლუპი.

თითოეულ ეპოქაში DataLoader იტვირთება;

tqdm აჩვენებს პროგრესს;

თითო batch-ზე ხდება:

1. მოდელის გამოტანა (forward),
2. Loss გამოთვლა,
3. Backpropagation,
4. gradient clipping (ნორმალიზაცია),
5. optimizer-ის ნაბიჯი.

Checkpoint შენახვა ყოველ 500 batch-ზე

```
if batches_done % save_every == 0:
   torch.save({...})
```

ინახავს:

1. მოდელის, ოპტიმიზატორის, scheduler-ის მდგომარეობას;
2. epoch-ს, batch-ს, და RNG-ებს;
3. sample-ის ინდექსების ზუსტ მიმდევრობას (epoch_indices).

ეს საშუალებას იძლევა მოდელი გაიგრძელოს სწავლებას ზუსტად იმავე ადგილიდან, იგივე randomness-ით.

ვალიდაციის ფაზა

თითო ეპოქის ბოლოს მოდელი გადადის eval რეჟიმში და ითვლება საშუალო validation loss.

Scheduler ამაზე რეაგირებს და საჭიროებისამებრ ამცირებს learning rate-ს.

საუკეთესო checkpoint-ის შენახვა

თუ validation loss უმჯობესდება:

```
torch.save({...}, best_path)
```

ინახავს როგორც "best" მოდელს.
თუ არა — ამცირებს patience მთვლელს;
თუ patience = 0 → early stopping.

ცხრილი 3. გადასაწვრთნელი პარამეტრების შეჯამება

ბლოკი	დანიშნულება
Prompt Variants	მრავალფეროვანი ინსტრუქციების სწავლა
Dataset	სურათისა და აღწერის ჩატვირთვა
Sampler	ზუსტი გაგრძელება შეწყვეტის შემდეგ
Model + Optimizer	სწავლების ბირთვი
Scheduler	სწავლის ტემპის ადაპტაცია
Checkpoints	შუალედური შენახვა და განგრძობადობა
Validation	პროგრესის შემოწმება
Early Stopping	ავტომატური გაჩერება გაუმჯობესების არქონისას

4.2 სინთეზირებული მოდელის აღწერა

იდეა: BLIP-2-ის ხედვის ნაწილი (Vision Encoder + Q-Former) აქცევს სურათს „კომპაქტურ“ ემბედინგებად. ეს ემბედინგები შემდეგ ფორმატდება ByT5-ის დამალულ სივრცეში (1536 ზომა) და წინ უსვამთ ტექსტურ პრომპტს, რათა ByT5-ის გენერატორმა (decoder-მა) დაამუშაოს და ააგოს სრული აღწერა.

ბლოკები:

1. Vision Encoder (BLIP-2) — იღებს სურათს და გამოაქვს ვიზუალური „ტოკენები“
2. Q-Former — სწავლადი query token-ებით იკვებება და „კითხულობს“ ხედვის ემბედინგებს, უბრუნებს K წერტილში შეკუმშულ სემანტიკას.
3. Projection (Linear→ReLU→Dropout) — Q-Former-ის გამოსვლას გადმოაქვს ByT5-ის ჰიდენ ზომაში = 1536.
4. ByT5 (T5ForConditionalGeneration, „google/byt5-base“)
    - Encoder stack გაყინულია (არ ვასწავლობთ).

- Decoder (გენერატორი) ვასწავლით—ის აგებს ქართულ კაპშენს.

რატომ?

Encoder გაყინულია, რადგან Q-Former + projection ისწავლოს სურათის „გადათარგმნა“ იმ სივრცეში, რასაც უკვე მიჩვეულია ByT5-ის encoder; თვითონ Encoder-ის გადააწყობა აღარ გვჭირდება.

Decoder ვასწავლოთ, რომ უკეთ წარმოთქვას ნათქვამი ქართულად.

კოდი—ხაზობრივი/ბლოკური ახსნა

```
import torch
import torch.nn as nn
from transformers import Blip2Model, T5ForConditionalGeneration
```

იტვირთება PyTorch და HuggingFace-ის ორი მოდული: Blip2Model (ხედვის ბლოკები + Q-Former) და T5ForConditionalGeneration (ტექსტის გენერატორი).

```
class Martha(nn.Module):
  def __init__(self):
    super().__init__()
```

ვქმნით PyTorch მოდულს Martha.

```
self.blip2 = Blip2Model.from_pretrained("Salesforce/blip2-opt-2.7b")
```

იტვირთება BLIP-2 ბაზური მოდელი (OPT-2.7B ვარიანტი). მნიშვნელოვანი: აქ ვიყენებთ Blip2Model-ს, ანუ ლინგვისტურ LM-ს არ ვტვირთავთ BLIP-2-დან; გვაინტერესებს მხოლოდ vision + Q-Former.

```
self.vision_encoder = self.blip2.vision_model
self.qformer = self.blip2.qformer
self.query_tokens = nn.Parameter(self.blip2.query_tokens.data.clone())
```

ხედვის მოდელი (Vision Transformer/EVA-სტილის backbone) და Q-Former ამოვიღეთ ცალკე ველებად.

query_tokens—BLIP-2-ის სწავადი „კითხვის ტოკენები“. ვკლონავთ და ვაცხადებთ ჩვენს ტრენირებად პარამეტრად, რომ დარეგულირდეს უშუალოდ ჩვენს ამოცანაზე.

```
# Project Q-Former output to ByT5 hidden size (1536)
self.qformer_proj = nn.Sequential(
    nn.Linear(self.qformer.config.hidden_size, 1536),
    nn.ReLU(),
    nn.Dropout(0.1)
)
```

Q-Former-ის გამოსულების ზომა ≠ ByT5-ის ჰიდენ ზომა (1536).

ეს პროექცია „გადააქართულებს“ (align) Q-Former ემბედინგებს byt5-ის სივრცეში:

1. Linear — განზომილების გასწორება;
2. ReLU — არახაზოვანი გარდაქმნა;
3. Dropout(0.1) — რეგულარიზაცია overfitting-ის საწინააღმდეგოდ.

```
# ByT5 decoder
self.decoder = T5ForConditionalGeneration.from_pretrained("google/byt5-base")
```

ვტვირთავთ ByT5-base-ს (ბაიტზე მომუშავე T5), რომელიც უკეთ მუშაობს მრავალ ენაზე, მათ შორის ქართულზე.

```
# Freeze vision encoder
for p in self.vision_encoder.parameters():
    p.requires_grad = False
for p in self.qformer.parameters():
    p.requires_grad = True
```

```
for p in self.decoder.encoder.parameters():
    p.requires_grad = False
self.query_tokens.requires_grad = True
```

1. Vision encoder ცივდება (არ ვასწავლით) — სურათის დაბალი დონის ფიჩერები
2. Q-Former ვასწავლით — მან უნდა „გაიგოს“, როგორ ამოიღოს სურათიდან რელევანტური სემანტიკა.
3. ByT5 Encoder stack იყინება — იდეა: სურათის ემბედინგები ჩვენ მოვარგოთ byt5-ს, არა პირიქით.
4. query_tokens — სწავლადი რჩება.

შენიშვნა: ByT5-ში embeddings ხშირად sharedა (encoder/decoder შორის). აქ გაყინულია decoder.encoder (Stack), მაგრამ შეიძლება embeddings მაინც დარჩეს ტრენირებადი, თუ არ გავყინავთ self.decoder.shared. თუ embeddings-ის გაყინვაც გინდა, დამატებით გაყინე self.decoder.shared.parameters().

forward(...) — სწავლების გზა

```
vision_out = self.vision_encoder(pixel_values=images).last_hidden_state
```

სურათები → ხედვის encoder → მივიღეთ ვიზუალური ტოკენების მიმდევრობა: [B, V, H_v].

```
qformer_out = self.qformer(
    query_embeds=self.query_tokens.expand(images.size(0), -1, -1),
    encoder_hidden_states=vision_out,
    encoder_attention_mask=torch.ones(vision_out.size()[:-1], dtype=torch.long, device=images.device)
).last_hidden_state
```

Q-Former იღებს:

- query_embeds — ერთი და იგივე სწავადი query ტოკენები, გაფართოებული batch-ზე (expand);
- encoder_hidden_states — ხედვის ტოკენები, რომელთაც Q-Former „კითხულობს“ (cross-attention);
- encoder_attention_mask=1 — ყველა ვიზუალური ტოკენი აქტიურია.

გამოდის შეკუმშული გამოსახულების ემბედინგები [B, Nq, H_q] (Nq — query ტოკენების რაოდენობა).

```
img_embed = self.qformer_proj(qformer_out)  # [B, Nq, 1536]
```

სურათის ემბედინგები დაგვყავს ByT5-ის ზომაზე — 1536.

```
input_embeds = self.decoder.encoder.embed_tokens(decoder_input_ids)  # [B, L, 1536]
```

პრომპტის ტოკენების ემბედინგები (სადაც decoder_input_ids სინამდვილეში encoder-ის შესატანია, ანუ პრომპტია).

```
inputs_embeds = torch.cat([img_embed, input_embeds], dim=1)

img_mask = torch.ones((images.size(0), img_embed.size(1)), dtype=torch.long, device=images.device)

attention_mask = torch.cat([img_mask, decoder_attention_mask], dim=1)
```

ვუკრავთ სურათის ემბედინგებს პრომპტის წინ (image-tokens + text-prompt).

ვაწყობთ შესაბამის attention mask-ს: სურათის ნაწილს მთელი ერთიანები აქვს, ტექსტს — tokenizer-ის ნიღაბი.

```
return self.decoder(
   inputs_embeds=inputs_embeds,
   attention_mask=attention_mask,
   labels=labels)
```

1. ახლა ეს ემბედინგები შედის ByT5 encoder-ში (რადგან inputs_embeds = encoder input).

2. labels — სამიზნე კაპშენის ტოკენებია; T5-ი თვითონ აკეთებს shift-ს decoder-ში.
3. დაბრუნებული ობიექტი მოიცავს loss-ს (გამოითვლება CrossEntropy-თ, სადაც padding = -100 იგნორირდება), logits და სხვ.

რატომ არის ასე აწყობილი?

1. Encoder გაყინულია → Q-Former + projection სწავლობს „სურათის ემბედინგების ზღაპრულ თარგმანს“ byt5-ის ტექსტურ სივრცეში.
2. Decoder სწავლობს გენერაციას ქართულად, ადამიანურ სტილში.

generate_caption(...) — ინფერენსი/გენერაცია

```
self.eval()
with torch.no_grad():
    vision_out = self.vision_encoder(pixel_values=image.to(device)).last_hidden_state
    qformer_out = self.qformer(...).last_hidden_state
    img_embed = self.qformer_proj(qformer_out)  # [1, Nq, 1536]
prompt_inputs = tokenizer(prompt, return_tensors="pt").to(device)
prompt_embeds = self.decoder.encoder.embed_tokens(prompt_inputs["input_ids"])
inputs_embeds = torch.cat([img_embed, prompt_embeds], dim=1)
attention_mask = torch.cat([
    torch.ones((1, img_embed.size(1)), dtype=torch.long, device=device),
    prompt_inputs["attention_mask"]
], dim=1)
```

1. ვამზადებთ პრომპტს (მაგ., „რა ხდება სურათზე?“).
2. კვლავ ვუკრავთ: [image tokens] + [prompt tokens] → encoder.

```
prompt_len = prompt_inputs["input_ids"].shape[1]
```

max_new_tokens = min(10 * prompt_len, 512)

გენერაციის ზედა ზღვარი—პროპორციულია პრომპტის სიგრძეზე (ზღვარი 512). პრაქტიკაში ეს იძლევა მოქნილ მაქსიმუმს.

generated_ids = self.decoder.generate(

inputs_embeds=inputs_embeds,

attention_mask=attention_mask,

max_new_tokens=max_new_tokens,

num_beams=5,

early_stopping=False,

length_penalty=0.9,

repetition_penalty=1.1)

1. Beam search (5 კაცი) უკეთესი სიგრძე/ხარისხის ბალანსისთვის;
2. length_penalty=0.9 — ოდნავ არსჯის გრძელს;
3. repetition_penalty=1.1 — აჩერებს თვითგამეორებას;
4. early_stopping=False — არ ჩერდეს მხოლოდ EOS-ზე პირველივე შანსზე (არჩეული სტილი).

return tokenizer.decode(generated_ids[0], skip_special_tokens=True)

ვაბრუნებთ დექოდებულ კაპშენს.

რატომ მუშაობს ეს დიზაინი

1. BLIP-2 + Q-Former: სურათის სემანტიკური „რეზიუმე“ მცირე რაოდენობის ტოკენად.
2. Projection → 1536: ზუსტად ByT5-ის დამალულ ზომაში მოხვედრა, რომ encoder-მა შეძლოს დამუშავება ზუსტად ისე, როგორც ტექსტურ ტოკენებს.

3. გაყინული encoder: სტაბილურობა + ნაკლები პარამეტრი → სწრაფი/მდგრადი სწავლა; სურათის მხარე „სწორდება“ byt5-ზე, არა პირიქით.
4. გენერაცია byt5-ის decoder-ით: ძლიერი, გრამატიკულად სწორი მეტყველება ქართულზე.

თავი V. შეფასების მეთოდოლოგია

პროექტის წარმატების გასაზომად გამოიყენება როგორც ავტომატური მეტრიკები, ასევე ადამიანური შეფასება. ასეთი კომბინირებული მიდგომა იძლევა როგორც რაოდენობრივ, ისე ხარისხობრივ სურათს მოდელის მუშაობის შესახებ.

5.1 ადამიანური შეფასება (Human Evaluation)

ქართულენოვანი სურათის აღწერის სისტემის შეფასებისას ერთ-ერთი მთავარი გამოწვევა ავტომატური მეტრიკების გამოყენებაა, განსაკუთრებით მაშინ, როდესაც ეს მეტრიკები ქართულის სრულ მხარდაჭერას არ ფლობენ. ერთი შეხედვით შეიძლება ჩანდეს, რომ ისეთი ფართოდ გამოყენებული ინსტრუმენტები, როგორიცაა BLEU, ROUGE, METEOR და CIDEr, უნივერსალურია და ნებისმიერ ენაზე გამოდგება. თუმცა პრაქტიკაში მათი გამოყენება ქართულზე სერიოზულ მეთოდოლოგიურ პრობლემებს წარმოშობს.

ეს მეტრიკები ძირითადად ინგლისურ ენაზეა ორიენტირებული და ეფუძნება სიტყვათა ზედაპირულ დამთხვევას, n-გრამების თანხვედრასა და წინასწარ განსაზღვრულ ლექსიკურ რესურსებს. ქართულ ენას კი ახასიათებს რთული მორფოლოგია, ბრუნვათა მრავალფეროვნება, ზმნის ფორმათა სირთულე და სიტყვათა შედარებით თავისუფალი რიგი. შედეგად, ერთი და იგივე მნიშვნელობა შეიძლება სხვადასხვა ფორმით გამოიხატოს, რაც ავტომატური მეტრიკისთვის სრულიად განსხვავებულ სიტყვებად აღიქმება.

მაგალითად, თუ საცნობარო აღწერაში წერია: „ქალი წითელ ქურთუკში დგას ქუჩაში“, ხოლო მოდელის მიერ გენერირებული ტექსტია: „ქალი წითელი ქურთუკით დგას ქუჩაზე“, სემანტიკური თვალსაზრისით ეს აღწერები თითქმის იდენტურია. თუმცა ავტომატური მეტრიკა დაინახავს განსხვავებულ ფორმებს — „ქურთუკში“ და „ქურთუკით“, „ქუჩაში“ და „ქუჩაზე“ — და ქულას შეამცირებს. ანალოგიურად,

სინონიმების შემთხვევაში („პარკი“ და „ბაღი“) ქულა კვლავ დაიკლებს, რადგან ქართულისთვის სპეციფიკური სინონიმთა ბაზა მეტრიკას არ გააჩნია.

ამგვარი შეფასება იწვევს სისტემატურ დამახინჯებას: მოდელი შეიძლება რეალურად ქმნიდეს სემანტიკურად სწორ და ბუნებრივ აღწერებს, მაგრამ მიიღოს ხელოვნურად დაბალი შედეგები. ეს თავის მხრივ საფრთხეს უქმნის კვლევის სანდოობას. მკვლევარმა შეიძლება მცდარი დასკვნა გამოიტანოს, რომ არქიტექტურა არაეფექტურია ან ტრენინგი არასაკმარისია, მაშინ როდესაც რეალური პრობლემა შეფასების ინსტრუმენტშია. შედეგად, მოდელის გაუმჯობესების პროცესიც არასწორი მიმართულებით წარიმართება.

ამ კონტექსტში ადამიანური შეფასება გარდაუვალ აუცილებლობად იქცევა. კვლევაში შერჩეულია 500 შემთხვევითი სურათი Validation/Test სეტიდან, რაც უზრუნველყოფს შედეგების სტატისტიკურ სანდოობას და ამცირებს შერჩევის მიკერძოებას. შეფასებაში მონაწილეობს 10 ნეიტრალური მონაწილე, რათა ინდივიდუალური სუბიექტუროობა დაბალანსდეს მრავალპერსპექტიული შეფასებით.

შეფასება ხორციელდება სამი ძირითადი კრიტერიუმის მიხედვით. პირველი არის სიზუსტე (Correctness) — აღწერს თუ არა გენერირებული ტექსტი რეალურად გამოსახულ ობიექტებსა და მოვლენებს, ხომ არ შეიცავს ჰალუცინაციებს ან ფაქტობრივ შეცდომებს. მეორე კრიტერიუმია სისუფთავე (Fluency) — რამდენად ბუნებრივად და გრამატიკულად გამართულად არის ჩამოყალიბებული ტექსტი ქართულ ენაზე. მესამე კომპონენტია სემანტიკური სიღრმე (Detail) — რამდენად ამომწურავია აღწერა, მოიცავს თუ არა მნიშვნელოვან დეტალებს, ურთიერთქმედებებსა და კონტექსტს.

თითოეული კრიტერიუმი ფასდება 1–5 ბალიანი Likert-ის შკალით, სადაც 1 აღნიშნავს ძალიან დაბალ ხარისხს, ხოლო 5 — ძალიან მაღალს. საბოლოო შედეგი გამოითვლება საშუალო მნიშვნელობის სახით (MOS — Mean Opinion Score), რაც იძლევა სტაბილურ და შედარებით ობიექტურ ინდიკატორს მოდელის ხარისხის შესახებ.

შეფასება:

1 – ძალიან ცუდი

2 – სუსტი

3 – საშუალო

4 – კარგი

5 – ძალიან კარგი

შემდეგ:

$$MOS = \frac{\sum_{i=1}^{N} scor\ e_i}{N}$$

სადაც:

- N = შეფასებების რაოდენობა
- საბოლოო ქულა = საშუალო მნიშვნელობა

ეს უზრუნველყოფს:

- სტაბილურობას
- სტატისტიკურ სანდოობას
- მოდელებს შორის ობიექტურ შედარებას

ამრიგად, თუ კვლევა დაეყრდნობოდა მხოლოდ ისეთ ავტომატურ მეტრიკებს, რომლებიც ქართულის მორფოლოგიურ და სემანტიკურ სპეციფიკას არ ითვალისწინებს, მიღებული შედეგები იქნებოდა არასამართლიანი და დამაბნეველი. ადამიანური შეფასება ამ შემთხვევაში არა უბრალოდ დამატებითი კომპონენტია, არამედ აუცილებელი მეთოდოლოგიური ნაბიჯი, რომელიც უზრუნველყოფს ქართული ენის კონტექსტში სურათის აღწერის სისტემის რეალური ხარისხის ადეკვატურ ასახვას.

დასკვნა

პროექტი „Martha“ წარმოადგენს პირველ მცდელობას საქართველოში, ვიზუალური გარემოს აღწერის სისტემა ქართულ ენაზე იქმნება თანამედროვე Encoder–Decoder არქიტექტურის საფუძველზე. პროექტის განხორციელების მეთოდოლოგია აერთიანებს:

1. მონაცემთა ფართო ბაზის შექმნას (300,000 სურათი და ქართული კაპშენი),
2. BLIP2 encoder-ის გამოყენებას ვიზუალური წარმოდგენების მისაღებად,
3. ByT5 decoder-ის ინტეგრაციას ტექსტის გენერაციისთვის, რომელიც ქართულ ენაზე მუშაობს,

4. Projection layer-ის მათემატიკურ სინქრონიზაციას, რათა ვიზუალური და ენობრივი სივრცეები თავსებადი იყოს,
5. რეგულარიზაციისა და ოპტიმიზაციის თანამედროვე ტექნიკებს, რომლებიც უზრუნველყოფენ სტაბილურ და ზუსტ ტრენინგს.

შედეგად ვიღებთ მოდელს, რომელსაც შეუძლია სურათების ქართულ ენაზე აღწერა, მაღალი სიზუსტით და ბუნებრივი ენობრივი სტილით.

1. მიღებული მოდელი არის blip2 ვიზუალურ-ენობრივი მოდელის გაუმჯობესებული ვერსია , ინოვაციური მიდგომის გზით, ორი მოდელის სინთეზით.
2. Martha მოდელის გადაწვრთვნა შესაძელებელია ნებისმიერ ენაზე, რომლის მხარდაჭერაც აქვს byt5 დიდ ენობრივ მოდელს, შესაბამის ენაზე მონაცემების შექმნის გზით , როგორც ეს მოხდა ქართული ენის შემთხვევაში. ეს ენებია :
    - Spanish
    - French
    - German
    - Russian
    - Chinese
    - Arabic
    - Portuguese
    - Armenian
    - Azerbaijani
    - Ukrainian
    - Turkish
    - Polish

პროექტის წარმატებით განხორციელება შექმნის პრეცედენტს ქართულ AI კვლევებში. მისი მნიშვნელობა მოიცავს:

1. ენობრივი ინკლუზიურობა – ქართული ენა პირველად იქნება წარმოდგენილი თანამედროვე ვიზუალურ–ენობრივ მოდელებში.

2. ინკლუზიური ტექნოლოგიები – მხედველობადაქვეითებული ადამიანებისთვის გარემოს აუდიო აღწერის შესაძლებლობა.
3. განათლება და მეცნიერება – ახალი ინსტრუმენტი, რომელიც გამოიყენება როგორც სასწავლო, ისე კვლევით პროცესებში.
4. ტურიზმი და კულტურა – საქართველოს კულტურული მემკვიდრეობის პოპულარიზაცია, ვიზუალური მასალების ქართულენოვანი აღწერებით.

მოსალოდნელი სამომავლო კვლევები

პროექტის დასრულების შემდეგ იგეგმება რამდენიმე მიმართულება:

1. მულტიმოდალური გაფართოება

   ვიდეოს აღწერის შესაძლებლობა (Video Captioning).

   აუდიოს ინტეგრაცია (Audio–Visual Captioning).

2. მულტილინგვური მხარდაჭერა

   მოდელის გაფართოება რეგიონულ ენებზე (სომხური, აზერბაიჯანული, ოსური).

   კოდ-სვიჩინგის მხარდაჭერა ქართულ–ინგლისური ტექსტებისთვის.

3. რეალურ დროში captioning

   დაბალი ლატენტურობის მოდელის ვერსია მობილური აპლიკაციებისთვის.

   ინტეგრაცია სმარტ სათვალეებში („ჭკვიანი სათვალე“ მხედველობადაქვეითებულთათვის).

4. RAG (Retrieval-Augmented Generation) ინტეგრაცია

   გენერირებული კაპშენების გამდიდრება გარე ცოდნის ბაზებით (მაგ. ტურისტული ობიექტების ისტორია). (Tsiramua,Meladze,Davitashvili,Bitmalkishev,Elbakidze/ჩირამუა,მელაძე,დავით აშვილი,ბიტმალკიშევ,ელბაქიძე. 2025)

5. მოდელის ოპტიმიზაცია

მოდელის შეკუმშვა (quantization, pruning) მობილურ და edge-დევაისებზე გასაშვებად.

ენერგოეფექტური სწავლება მცირე რესურსების მქონე ლაბორატორიებისთვის.

დანართი 1. Martha მოდელის კოდი

```
# captioner.py
%%writefile captioner.py
import torch
import torch.nn as nn
from transformers import Blip2Model, T5ForConditionalGeneration


class Martha(nn.Module):
  def __init__(self):
    super().__init__()
    self.blip2 = Blip2Model.from_pretrained("Salesforce/blip2-opt-2.7b")
    self.vision_encoder = self.blip2.vision_model
    self.qformer = self.blip2.qformer
```

```
    self.query_tokens = nn.Parameter(self.blip2.query_tokens.data.clone())

    # Project Q-Former output to ByT5 hidden size (1536)
    self.qformer_proj = nn.Sequential(
      nn.Linear(self.qformer.config.hidden_size, 1536),
      nn.ReLU(),
      nn.Dropout(0.1))

    # ByT5 decoder
    self.decoder = T5ForConditionalGeneration.from_pretrained("google/byt5-base")


    # Freeze vision encoder
    for p in self.vision_encoder.parameters():
      p.requires_grad = False
    for p in self.qformer.parameters():
      p.requires_grad = True
    for p in self.decoder.encoder.parameters():
      p.requires_grad = False
    self.query_tokens.requires_grad = True

  def forward(self, images, decoder_input_ids, decoder_attention_mask, labels=None):
    vision_out = self.vision_encoder(pixel_values=images).last_hidden_state

    qformer_out = self.qformer(
      query_embeds=self.query_tokens.expand(images.size(0), -1, -1),
      encoder_hidden_states=vision_out,
      encoder_attention_mask=torch.ones(vision_out.size()[:-1], dtype=torch.long,
device=images.device)
    ).last_hidden_state

    img_embed = self.qformer_proj(qformer_out)  # [B, N, 1536]
    input_embeds = self.decoder.encoder.embed_tokens(decoder_input_ids)  # [B, L,
1536]

    inputs_embeds = torch.cat([img_embed, input_embeds], dim=1)
    img_mask = torch.ones((images.size(0), img_embed.size(1)), dtype=torch.long,
device=images.device)
    attention_mask = torch.cat([img_mask, decoder_attention_mask], dim=1)

    return self.decoder(
      inputs_embeds=inputs_embeds,
      attention_mask=attention_mask,
      labels=labels
    )

  def generate_caption(self, image, prompt, tokenizer, device="cuda"):
```

```
    self.eval()
    with torch.no_grad():
      vision_out = self.vision_encoder(pixel_values=image.to(device)).last_hidden_state
      qformer_out = self.qformer(
        query_embeds=self.query_tokens.expand(image.size(0), -1, -1),
        encoder_hidden_states=vision_out,
        encoder_attention_mask=torch.ones(vision_out.size()[:-1], dtype=torch.long,
device=device)
      ).last_hidden_state

      img_embed = self.qformer_proj(qformer_out)  # [1, N, 1536]

    prompt_inputs = tokenizer(prompt, return_tensors="pt").to(device)
    prompt_embeds =
self.decoder.encoder.embed_tokens(prompt_inputs["input_ids"])
    inputs_embeds = torch.cat([img_embed, prompt_embeds], dim=1)
    attention_mask = torch.cat([
      torch.ones((1, img_embed.size(1)), dtype=torch.long, device=device),
      prompt_inputs["attention_mask"]
    ], dim=1)

    prompt_len = prompt_inputs["input_ids"].shape[1]
    max_new_tokens = min(10 * prompt_len, 512)

    generated_ids = self.decoder.generate(
      inputs_embeds=inputs_embeds,
      attention_mask=attention_mask,
      max_new_tokens=max_new_tokens,
      num_beams=5,
      early_stopping=False,
      length_penalty=0.9,
      repetition_penalty=1.1
    )

    return tokenizer.decode(generated_ids[0], skip_special_tokens=True)
```

ინტერფეისი:

```
# inference.py
%%writefile inference.py

import torch
from PIL import Image
from transformers import AutoTokenizer, Blip2Processor
from captioner import Martha

device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
# ------------------ Load Tokenizer and Processor ------------------
tokenizer = AutoTokenizer.from_pretrained("google/byt5-base")
processor = Blip2Processor.from_pretrained("Salesforce/blip2-opt-2.7b")

# ------------------ Initialize Model ------------------
model = Martha().to(device)

# ------------------ Load Checkpoint ------------------
checkpoint_path = "drive/MyDrive/SavedModels/martha_checkpoint.pth"
checkpoint = torch.load(checkpoint_path, map_location=device)

model.load_state_dict(checkpoint["model_state"], strict=False)

model.eval()

# ------------------ Load and Preprocess Image ------------------
image_path = "drive/MyDrive/SavedModels/bird-8788491_1280.jpg"
image = Image.open(image_path).convert("RGB")
pixel_values = processor(images=image, return_tensors="pt")["pixel_values"].to(device)

# ------------------ Generate Caption ------------------
prompt = "რა ჩანს სურათში?"
caption = model.generate_caption(pixel_values, prompt, tokenizer, device=device)
print("🖼️ Generated Caption:", caption)
```

გადაწვრთვნის სკრიპტი:

```
# train.py
%%writefile train.py

import os
import re
import sys
import csv
import torch
import random
import numpy as np
import pandas as pd
from torch.utils.data import Dataset, DataLoader, Sampler
from transformers import AutoTokenizer
from captioner import Martha
from tqdm import tqdm

# ------------------ Prompts ------------------
prompt_variants = [
    "რა ხდება სურათზე?", "რა ჩანს სურათში?", "რას ხედავ სურათზე?",
    "დაწერე, რას ხედავ სურათში?", "დააკვირდი სურათის დეტალებს და აღწერე",
    "სურათის მოკლე აღწერა მომეცი", "რა ხდება სურათში",
```

```
    "შეაფასე სურათი მოკლედ", "რა ჩანს სურათის კადრში?",
    "გაახმოვანე სურათი სიტყვებით", "რას შეიცავს ეს სურათი?",
    "სურათის შინაარსი მომიყევი", "რას გეუბნება სურათი?",
    "შეაჯამე სურათის შინაარსი", "აღწერე რასაც ხედავ"
]

csv.field_size_limit(sys.maxsize)
device = "cuda" if torch.cuda.is_available() else "cpu"

# ------------------ Dataset ------------------
class ImageCaptionDataset(Dataset):
    def __init__(self, csv_path, tokenizer, split="train", prompt_variants=prompt_variants):
        df = pd.read_csv(csv_path)
        self.data = df[df["split"] == split].to_dict("records")
        self.tokenizer = tokenizer
        self.prompt_variants = prompt_variants
        print(f"📦 {split.upper()} set: {len(self.data)} tensors available")

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        row = self.data[idx]
        tensor_path = row["tensor_path"]
        image_tensor = torch.load(tensor_path, weights_only=False)

        caption = row["caption_ka"]
        prompt = random.choice(self.prompt_variants)

        prompt_tok = self.tokenizer(prompt, return_tensors="pt",
                                    padding="max_length", truncation=True, max_length=31)
        caption_tok = self.tokenizer(caption, return_tensors="pt",
                                     padding="max_length", truncation=True, max_length=480)

        labels = caption_tok["input_ids"].squeeze(0)
        labels[labels == self.tokenizer.pad_token_id] = -100

        return {
            "image": image_tensor,
            "decoder_input_ids": prompt_tok["input_ids"].squeeze(0),
            "decoder_attention_mask": prompt_tok["attention_mask"].squeeze(0),
            "labels": labels
        }

# ------------------ Paths ------------------
csv_path = "drive/MyDrive/image-captioning/georgian_captions_local_split.csv"
checkpoint_dir = "drive/MyDrive/SavedModels/regular"
```

```
best_dir = "drive/MyDrive/SavedModels/best"
os.makedirs(checkpoint_dir, exist_ok=True)
os.makedirs(best_dir, exist_ok=True)

# ------------------ Tokenizer ------------------
tokenizer = AutoTokenizer.from_pretrained("google/byt5-base")

# ------------------ Dataset ------------------
train_dataset = ImageCaptionDataset(csv_path, tokenizer, split="train")
val_dataset   = ImageCaptionDataset(csv_path, tokenizer, split="val")

# ------------------ FixedOrderSampler ------------------
class FixedOrderSampler(Sampler[int]):
  """Yields a fixed list of indices starting from an offset (in samples)."""
  def __init__(self, indices, offset=0):
    self.indices = list(map(int, indices))
    self.offset = int(offset)
    if self.offset < 0 or self.offset > len(self.indices):
      raise ValueError(f"offset {self.offset} out of range 0..{len(self.indices)}")

  def __iter__(self):
    for i in range(self.offset, len(self.indices)):
      yield self.indices[i]

  def __len__(self):
    return len(self.indices) - self.offset

# ------------------ Model / Optim / Sched ------------------
model = Martha().to(device)
optimizer = torch.optim.AdamW([
  {"params": model.qformer_proj.parameters(), "lr": 1e-5},
  {"params": model.decoder.parameters(), "lr": 5e-5},
  {"params": model.qformer.parameters(), "lr": 1e-5},
  {"params": [model.query_tokens], "lr": 1e-5}
], weight_decay=0.01)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
  optimizer, mode='min', patience=2, factor=0.5
)

# ------------------ Helpers ------------------
def cleanup_checkpoints(directory, keep_last=3):
  ckpts = sorted(
    [os.path.join(directory, f) for f in os.listdir(directory) if f.endswith(".pth")],
    key=os.path.getmtime, reverse=True
  )
  for old_ckpt in ckpts[keep_last:]:
    try:
```

```
            os.remove(old_ckpt)
            print(f"🗑️ Deleted old checkpoint: {os.path.basename(old_ckpt)}")
        except Exception as e:
            print(f"⚠️ Could not delete {old_ckpt}: {e}")

def latest_checkpoint(path):
    files = [f for f in os.listdir(path) if f.endswith(".pth")]
    if not files:
        return None
    return max(files, key=lambda f: os.path.getmtime(os.path.join(path, f)))

def make_epoch_indices(n_items, base_seed, epoch):
    """Deterministic per-epoch shuffle (reproducible)."""
    g = torch.Generator()
    g.manual_seed(int(base_seed) + int(epoch))
    return torch.randperm(n_items, generator=g).tolist()

# ------------------ Config / Resume state ------------------
epochs = 100
patience = 10
save_every = 500        # every 500 batches
batch_size = 16
base_shuffle_seed = 123456789

start_epoch   = 0       # epoch number stored in ckpt (0-based)
resume_batch  = 0       # number of batches already completed in THIS epoch
best_val_loss = float("inf")
epoch_indices = None    # full sample order for this epoch (list of indices)

# Try to resume
ckpt_file = latest_checkpoint(checkpoint_dir)
if ckpt_file:
    path = os.path.join(checkpoint_dir, ckpt_file)
    print(f"🔁 Resuming from {path}")
    ckpt = torch.load(path, map_location="cpu", weights_only=False)

    model.load_state_dict(ckpt["model_state"])
    optimizer.load_state_dict(ckpt["optimizer_state"])
    scheduler.load_state_dict(ckpt["scheduler_state"])

    start_epoch   = int(ckpt.get("epoch", 0))
    resume_batch  = int(ckpt.get("resume_batch", 0))  # <-- batches completed
    best_val_loss = float(ckpt.get("best_val_loss", float("inf")))
    base_shuffle_seed = int(ckpt.get("base_shuffle_seed", base_shuffle_seed))

    # Restore RNG states (strict reproducibility)
    if "torch_rng_state" in ckpt:
```

```
      torch.set_rng_state(ckpt["torch_rng_state"])
    if "cuda_rng_state" in ckpt and torch.cuda.is_available() and ckpt["cuda_rng_state"] is
not None:
      torch.cuda.set_rng_state_all(ckpt["cuda_rng_state"])
    if "numpy_rng_state" in ckpt:
      np.random.set_state(ckpt["numpy_rng_state"])
    if "random_rng_state" in ckpt:
      random.setstate(ckpt["random_rng_state"])

    # Epoch order: use saved order if present; otherwise recreate deterministically
    if "epoch_indices" in ckpt and ckpt["epoch_indices"] is not None:
      epoch_indices = list(map(int, ckpt["epoch_indices"]))
    else:
      epoch_indices = make_epoch_indices(len(train_dataset), base_shuffle_seed,
start_epoch)

    total_batches = (len(epoch_indices) + batch_size - 1) // batch_size
    # ------------------ Smart Resume Decision ------------------
    # Check if there's already a validated checkpoint for this epoch
    best_files = [f.strip().lower() for f in os.listdir(best_dir)]
    target_name = f"martha_best_epoch{start_epoch}.pth".lower()

    validated_exists = target_name in best_files

    if validated_exists:
      print(f"🏁 Detected validated checkpoint for epoch {start_epoch} in BEST folder →
Skipping to next epoch.")
      start_epoch += 1
      resume_batch = 0
    elif resume_batch >= total_batches:
      print(f"🧭 Completed epoch {start_epoch}
(batches={resume_batch}/{total_batches}) → Moving to next epoch.")
      start_epoch += 1
      resume_batch = 0
    elif resume_batch == 0:
      print(f"🧭 Validation already finished for epoch {start_epoch} → Moving to next
epoch.")
      start_epoch += 1
    else:
      print(f"🔚 Resume info: epoch={start_epoch}, starting from batch
{resume_batch+1}/{total_batches}")

else:
    print("✅ Model initialized from scratch.")
    epoch_indices = make_epoch_indices(len(train_dataset), base_shuffle_seed,
start_epoch)
```

```
# ------------------ DataLoader builders ------------------
def make_train_loader(indices, resume_batch, batch_size):
    """Start exactly at the next batch using a sample offset = resume_batch *
batch_size."""
    sample_offset = resume_batch * batch_size
    sampler = FixedOrderSampler(indices, offset=sample_offset)
    return DataLoader(
        train_dataset,
        batch_size=batch_size,
        sampler=sampler,
        shuffle=False,
        num_workers=4,
        pin_memory=True, )

val_loader = DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=4,
    pin_memory=True,


)

# ------------------ Training loop ------------------
epochs_no_improve = 0

for epoch in range(start_epoch, epochs):
    model.train()
    total_batches = (len(epoch_indices) + batch_size - 1) // batch_size

    if epoch == start_epoch and resume_batch > 0:
        print(f"\n 🌿  --- Resuming Epoch {epoch} from batch
{resume_batch+1}/{total_batches} ---")
    else:
        print(f"\n 🌿  --- Starting Epoch {epoch} ---")

    train_loader = make_train_loader(epoch_indices, resume_batch, batch_size)
    pbar = tqdm(
        total=total_batches,
        initial=resume_batch,
        desc=f" 🚂 Training (Epoch {epoch})",
        unit="batch",
    )

    batches_done = resume_batch  # completed batches in this epoch
```

```
  for batch in train_loader:
    images = batch["image"].to(device)
    input_ids = batch["decoder_input_ids"].to(device)
    attention_mask = batch["decoder_attention_mask"].to(device)
    labels = batch["labels"].to(device)

    outputs = model(
      images,
      decoder_input_ids=input_ids,
      decoder_attention_mask=attention_mask,
      labels=labels
    )
    loss = outputs.loss

    optimizer.zero_grad()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
    optimizer.step()

    batches_done += 1  # we just finished this batch

    #  Save every 500 *batches* completed
    if batches_done % save_every == 0:
      ckpt_name = f"martha_epoch{epoch}_step{batches_done}.pth"
      ckpt_path = os.path.join(checkpoint_dir, ckpt_name)
      torch.save({
        "epoch": epoch,
        "resume_batch": batches_done,    # <-- batches completed
        "model_state": model.state_dict(),
        "optimizer_state": optimizer.state_dict(),
        "scheduler_state": scheduler.state_dict(),
        "best_val_loss": best_val_loss,
        "base_shuffle_seed": base_shuffle_seed,
        "epoch_indices": epoch_indices,   # full order → exact resume
        # RNG states for strict reproducibility
        "torch_rng_state": torch.get_rng_state(),
        "cuda_rng_state": torch.cuda.get_rng_state_all() if torch.cuda.is_available() else
None,
        "numpy_rng_state": np.random.get_state(),
        "random_rng_state": random.getstate()
      }, ckpt_path)
      pbar.write(f"💾 Saved checkpoint: {ckpt_name}")
      cleanup_checkpoints(checkpoint_dir, keep_last=5)

    pbar.update(1)
    pbar.set_postfix(loss=f"{loss.item():.4f}")
```

```
    pbar.close()

    # Finished ALL training batches for this epoch → reset for next
    resume_batch = 0
    # Prepare next epoch's deterministic order
    epoch_indices = make_epoch_indices(len(train_dataset), base_shuffle_seed, epoch
+ 1)

    # ------------------ Validation ------------------
    print(f"\n🔍 Starting validation after Epoch {epoch}...")
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for batch in val_loader:
            images = batch["image"].to(device)
            input_ids = batch["decoder_input_ids"].to(device)
            attention_mask = batch["decoder_attention_mask"].to(device)
            labels = batch["labels"].to(device)

            outputs = model(
                images,
                decoder_input_ids=input_ids,
                decoder_attention_mask=attention_mask,
                labels=labels
            )
            val_loss += outputs.loss.item()

    avg_val_loss = val_loss / len(val_loader)
    scheduler.step(avg_val_loss)
    print(f"✅ Validation Loss: {avg_val_loss:.4f}")

    # Save best snapshot (end-of-epoch)
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        best_name = f"martha_best_epoch{epoch}.pth"
        best_path = os.path.join(best_dir, best_name)
        print(f"⏳ Validation improved. Saving best checkpoint: {best_name}")
        torch.save({
            "epoch": epoch,
            "resume_batch": 0,   # next epoch starts at batch 0
            "model_state": model.state_dict(),
            "optimizer_state": optimizer.state_dict(),
            "scheduler_state": scheduler.state_dict(),
            "best_val_loss": best_val_loss,
            "base_shuffle_seed": base_shuffle_seed,
            "epoch_indices": epoch_indices,  # you may store next epoch's order too
            # RNG states
```

```
        "torch_rng_state": torch.get_rng_state(),
        "cuda_rng_state": torch.cuda.get_rng_state_all() if torch.cuda.is_available() else
None,
        "numpy_rng_state": np.random.get_state(),
        "random_rng_state": random.getstate()
      }, best_path)
    else:
      patience -= 1
      print(f" ⚠️ No improvement. Patience left: {patience}")
      if patience <= 0:
        print(" ⛔ Early stopping triggered!")
        break

print("\n ✅ Training completed and model saved.")
```

ნაშრომის ფარგლებში გამოქვეყნებული პუბლიკაციები და ჩატარებული კონფერენციები

1. David Bitmalkishev, Sergo Tsiramua, Hamlet Meladze, Tinatin Davitashvili. Analyzing Image Patterns and Generating Text: Advances in Multilingual Vision Language Transformers. Workshop CSIT-2025 on "Large Digital Models and Specific Pattern Analyses", the Institute for Informatics and Automation Problems, Yerevan, May 30-31, 2025.

2. Davit Bitmalkishev, Sergo Tsiramua, Hamlet Meladze, Tinatin Davitashvili, Tatia Elbakidze. Question-Answering System Bazed on AI and NLP Models. Proceedings of the XV Scientific Conference of the Union of Mathematicians of Georgia, Batumi, 1-6 September, 2025. https://gmu.gtu.ge/conferences/wp-content/uploads/2025/08/Conference_GMU_2025.pdf

3. Bitmalkishev Davit, Tsiramua Sergo, Meladze Hamlet, Davitashvili Tinatin, Elbakidze Tatia. AI and NLP Models for Q&A in Georgian. Proceedings of the 15th International Conference on Computer Science and Information Technologies CSIT 2025, Erevan, 2025. https://doi.org/10.51408/csit2025_11

4. Bitmalkishev Davit. Design and Training of a Georgian Vision-to-Text Model Using ViT and ByT5. Proceedings of the South Caucasus Congerence in Artificial Inteligence, SCCAI2025, Tbilisi, 2025.

5. Sergo Tsiramua, Hamlet Meladze, Davit Bitmalkishev. Analyzing Image Patterns and Generating Text: Advances in Multilingual Vision-Language Transformers. Journal “Pattern Recognition and Image Analysis. Advances in Mathematical Theory and Applications” issue 4, volume 35, 2025.

გამოყენებული ლიტერატურა

Alec Radford et al. Learning Transferable Visual Models From Natural Language Supervision. ICML, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

Alex Wang et al. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. 2018.

Alex Wang et al. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. 2019.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In CVPR, 2015.

Anna Rohrbach, Lisa Anne Hendricks, Kaylee Burns, Trevor Darrell, and Kate Saenko. Object hallucination in image captioning. arXiv preprint arXiv:1809.02156, 2018.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do CIFAR-10 classifiers generalize to CIFAR-10? arXiv:1806.00451, 2018.

Chao Jia et al. Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision. 2021.

Christoph Schuhmann et al. LAION-400M: Open Dataset of CLIP-Filtered 400 Million Image-Text Pairs. 2021.

Colin Raffel et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. JMLR, 2020.

Dim P Papadopoulos, Jasper RR Uijlings, Frank Keller, and Vittorio Ferrari. We don't need no bounding-boxes: Training object class detectors using only human verification. In CVPR, 2016.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-okvqa: A benchmark for visual question answering using world knowledge. In European Conference on Computer Vision, pp. 146–162. Springer, 2022.

Edward J. Hu et al. LoRA: Low-Rank Adaptation of Large Language Models. 2021.

Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. Advances in Computer Vision and Pattern Recognition, 2017.

Girish Kulkarni et al. Babytalk: Understanding and generating simple image descriptions. IEEE PAMI, 35(12):2891–2903, 2013.

Grossi, E., & Buscema, M. Introduction to artificial neural networks. European Journal of Gastroenterology & Hepatology, 2008.

Hao Tan & Mohit Bansal. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. EMNLP, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothee´ Lacroix, Baptiste Roziere, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023.

Hyung Won Chung et al. Scaling Instruction-Finetuned Language Models. 2022.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. Transactions on Machine Learning Research, 2022.

Jean-Baptiste Alayrac et al. Flamingo: a Visual Language Model for Few-Shot Learning. 2022.

Jeffrey Donahue et al. Long-term recurrent convolutional networks for visual recognition and description. CVPR, 2015.

Jonathan Huang et al. Speed/accuracy trade-offs for modern convolutional object detectors. CVPR, 2017.

Jun Chen, Han Guo, Kai Yi, Boyang Li, and Mohamed Elhoseiny. Visualgpt: Data-efficient adaptation of pretrained language models for image captioning. In CVPR, 2022.

Junhua Mao, Jiajing Xu, Kevin Jing, and Alan L Yuille. Training and evaluating multimodal word embeddings with large-scale web annotated images. In NIPS, 2016.

Junnan Li et al. Align before Fuse: Vision and Language Representation Learning with Momentum Distillation. 2021.

Junnan Li et al. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. 2022.

Junnan Li et al. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. 2023.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In ICML, 2015.

Kenneth Tran, Xiaodong He, Lei Zhang, Jian Sun, Cornelia Carapcea, Chris Thrasher, Chris Buehler, and Chris Sienkiewicz. Rich Image Captioning in the Wild. In CVPR Workshops, 2016.

Linting Xue et al. ByT5: Towards a token-free future with pre-trained byte-to-byte models. 2021.

Maria Tsimpoukelli, Jacob L Menick, Serkan Cabi, SM Eslami, Oriol Vinyals, and Felix Hill. Multimodal few-shot learning with frozen language models. Advances in Neural Information Processing Systems, 34:200–212, 2021.

Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. Journal of Artificial Intelligence Research, 2013.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In CVPR, 2015.

Peng Wang et al. OFA: Unifying Architectures, Tasks, and Modalities Through a Simple Sequence-to-Sequence Learning Framework. 2022.

Peng Wang et al. SimVLM: Simple Visual Language Model Pretraining with Weak Supervision. 2021.

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. SPICE: Semantic Propositional Image Caption Evaluation. In ECCV, 2016.

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Guided open vocabulary image captioning with constrained beam search. In EMNLP, 2017.

Peter Anderson, Stephen Gould, and Mark Johnson. Partially-supervised image captioning. In NIPS, 2018.

Piyush Sharma et al. Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning. ACL, 2018.

Ranjay Krishna et al. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. IJCV, 2017.

Sebastian Gehrmann et al. The GEM Benchmark: Natural Language Generation, its Evaluation and Metrics. 2021.

Shaoqing Ren et al. Faster R-CNN: Towards real-time object detection with region proposal networks. NIPS, 2015.

Soravit Changpinyo et al. Conceptual 12M: Pushing Web-Scale Image-Text Pre-Training To Recognize Long-Tail Visual Concepts. CVPR, 2021.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In CVPR, 2017.

Tsung-Yi Lin et al. Microsoft COCO: Common Objects in Context. ECCV, 2014.

Vicente Ordonez, Girish Kulkarni, and Tamara L. Berg. Im2Text: Describing Images Using 1 Million Captioned Photographs. NeurIPS, 2011.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90% chatgpt quality, March 2023.

Xi Chen et al. PaLI: A Jointly-Scaled Multilingual Language-Image Model. 2022.

Xiaohua Zhai et al. LiT: Zero-Shot Transfer with Locked-Image Text Tuning. CVPR, 2022.

Yuxin Fang et al. EVA: Exploring the Limits of Masked Visual Representation Learning at Scale. 2022.