Faculty of Computers &
Artificial Intelligence

Benha University

Automatic Grading For Essay Questions Using DL Models

A senior project submitted in partial fulfillment of the requirements
For the degree of Bachelor of Computers and Artificial Intelligence.

AI Departement,

Project Team

1- Mohamed Ahmed Mohamed Emam
2- Mohamed Bahgat Abd_Elkali Biomy
3- Abdelmonem Mansour Abdelmonem Ahmed
4- Mariem Ahmed Ebrahim Mohammed
5- Ehab AbdelAzim Abdel Hamid
6- Youssef sherif Mohamed salah

Under Supervision of

Dr. Mustafa Abdel Salam

En. Hosam Faker

June 2024

ACKNOWLEDGEMENT

وَمَا تَوْفِيقِي إِلَّا بِاللَّهِ عَلَيْهِ تَوَكَّلْتُ وَإِلَيْهِ أُنِيبُ.

We gratefully acknowledge the assistance, cooperation, guidance and clarifications provided by our guide during the development of the Online Exams System website.

Our extreme gratitude to **Dr. Mustafa Abdel Salam** who guided us throughout the project. Without his willing disposition, spirit of accommodation, frankness, timely clarification and above all faith in us, this project could not have been completed in due time. His readiness to discuss all important matters at work deserves special attention.

We also want to thank the whole college faculty for their cooperation and important support.

DECLARATION

We hereby certify that this material, which we now submit for assessment on the program of study leading to the award of Bachelor of Computers and Artificial Intelligence in (Artificial intelligence) is entirely our own work, that we have exercised reasonable care to ensure that the work is original, and does not to the best of our knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of our work.

Signed: *Mohamed Emam*
Signed: *Mohamed Bahgat*
Signed: *Abdelmonem Mansour*
Signed: *Mariem Ahmed*
Signed: *Ehab AbdelAzim*
Signed: *Youssef sherif*

Date: 24, June 2024

ABSTRACT

Online exams have become increasingly prevalent in educational institutions, providing a convenient and flexible way for students to demonstrate their knowledge and skills. However, the traditional exam format has several limitations that can be addressed through the incorporation of autocorrect and Language Model-based approaches.

With the advancement of technology, there is a growing need to improve the traditional exam format and incorporate innovative solutions to enhance the exam experience.

One area of improvement is the implementation of autocorrect and Language Model-based systems (LLMs) in online exams. These technologies can help address common challenges in exams such as time constraints, fairness, and accuracy of grading.

By incorporating autocorrect and LLMs into online exams, educational institutions can enhance the accuracy, efficiency, and effectiveness of the assessment process, providing a more valuable and supportive experience for both students and educators. This model help us ensuring that students' answers are accurately assessed.

In this document, we will explore the benefits of incorporating autocorrect and LLMs in online exams and how they can revolutionize the way exams are conducted and evaluated.

الملخص

تحولت الامتحانات عبر الإنترنت إلى شيء شائع في المؤسسات التعليمية، حيث توفر وسيلة مريحة ومرنة للطلاب لإظهار معرفتهم ومهاراتهم ومع ذلك، يعاني الشكل التقليدي للامتحان من عدة قيود يمكن التغلب عليها من خلال التصحيح التلقائي المعتمد علي النماذج اللغويه العملاقه .

مع تقدم التكنولوجيا، هناك حاجة متزايدة لتحسين التصحيح التقليدي للامتحان ودمج حلاً مبتكرًا لتعزيز تجربة الامتحان والتقليل من الاخطاء الناتجه من التصحيح البشري .

واحده مهمه من هذه التحسينات التي نحتاجها للأنظمه هي الاعتماد علي النماذج اللغويه في التصحيح التلقائي للامتحنات وذالك من خلال فهم المعني الحقيقي وراء كل اجابه ومقارناتها بالاجابات الصحيحه من خلال اعطاء الموديل مجموعه من الاجابات الصحيحه علي اشكال مختلفه وبدرجات متفاوته لفهم طبيعه الأسئله اكثر كما تم اضافه موديل للتعرف علي الرسومات وتصحيحها بالمقارنه بالاجابه النموزجيه للرسمه من خلال تقنيات النماذج التوليديه العملاقه .

من خلال تطبيق التصحيح التلقائي للأسئله بالستخدام النماذج اللغويه العملاقه علي الامتحنات عبر الانترنت يمكن للمؤسسات التعليمية تعزيز دقة وكفاءة وفعالية عملية التقييم، مما يوفر تجربة أكثر قيمة ودعمًا للطلاب والمعلمين . يمكن لهذا النموذج أن يساعد في ضمان تقييم إجابات الطلاب بدقة .

في هذا الملف سوف نتعرف علي كيفيه دمج نظام التصحيح الالكتروني بالستخدام النماذج اللغويه العملاقه مع نظم الامتحانات التقليديه للحصول علي دقه اعلي وتوفير للوقت والجهد وتوفير ايضا التكلفه . لقد حاولنا تطبيق اكتثر من 20 نموذج لغويه عملاق ب احجام مختلفه وانواع مختلفه علي داتا عربي وانكليزيه وقمنا بتجارب كثيرة جدا حتي وصلنا ل افضل اداء لهذه النماذج .

Table of Contents

Tables

Table of figures

Chapter 1

INTRODUCTION

Online Exams are being launched because of a need for a destination that is beneficial for both institutes and students. In Arabic or English exams especially essay questions, evaluating student answers and determining their similarity to the reference answer can be a time-consuming task.

Our project of creating an automated essay exam website endeavors to revolutionize the assessment paradigm through the integration of cutting-edge advancements in natural language processing and artificial intelligence.

With our site, educational organizations can register and host online exams. Students can take exams and view their results. This site is an attempt to remove the existing flaws in the manual system of conducting exams. With our automated system, students can submit their essays and receive instant corrections, allowing them to identify areas of improvement and enhance their writing skills. Our website will utilize a large language model that has been trained on a vast corpus of texts, enabling it to accurately analyze and correct essays. Leveraging Language Learning Models (LLMs) to automatically calculate the similarity between the reference answer and student answers can streamline this process.

We believe that our automated essay exam website will greatly benefit students by offering them a convenient and reliable tool for improving their writing skills. we aim to empower students to take control of their learning and excel in their exams.

BACKGROUND:

Language models (LLMs) are a type of artificial intelligence (AI) model trained on massive amounts of text data. They excel at understanding and generating human language, predicting the likelihood of word sequences, and even creating new text based on a given input. Common LLM architectures like GPT-3 and Jurassic-1 Jumbo utilize the powerful self-attention mechanism from Transformers to achieve these capabilities.

However, LLMs primarily focus on textual data. To bridge the gap between language and visual information, Vision Language Models (VLMs) have emerged. VLMs are a specialized type of AI model trained on both text and image data. This allows them to understand the relationships between visual content and textual descriptions.

Here's how LLMs and VLMs can work together for image comparison:

- **LLMs for Textual Analysis:** LLMs can analyze text descriptions associated with images, extracting key concepts and identifying similarities or differences.
- **VLMs for Visual Understanding:** VLMs can process the visual content of images, identifying objects, scenes, and their relationships.
- **Comparison and Reasoning:** By combining the insights from LLMs and VLMs, the system can compare images based on both their textual descriptions and visual content.

This integration allows for more comprehensive and nuanced image comparisons compared to relying solely on text or visual analysis alone.

OVERVIEW :

This project assesses students by conducting online objective tests. The tests would be highly customizable. This project will enable educational institutes to

conduct tests and have automated checking of answers based on the response by the candidates.

The project allows faculties to create their own tests. It would enable educational institutes to perform tests, quiz. It asks faculty to create his/her set of questions. Further the tests are associated with specific groups so that only associated students can appear for the test. The result of the response would be available to the faculty of the question set. This project would be helpful for creating practice tests, say for educational institutes and as a feedback form.

The online test created for taking online test has following stages:

- Login
- Test
- Result

Login:

There is a quality login window because this is more secure than other login forms as in a normal login window there are multiple logins available so that more than one person can access to test with there individual login. But in this project there is only one login id i.e. administrator id and password by which a person enter the site. Hence it is more secure and reliable than previously used on-line test simulators.

TEST:

Test page is the most creative and important page in this project. It consists of 2 modules namely:

- level selection
- Image compering
- language selection:-

  Arabic or English

- Utilities:- It includes Skip and come back to the question afterwards if needed. can either attempt or change the answer of the already attempted question

Purpose:

fulfills the requirements of Educational organizations to conduct the exams online. They do not have to go to any software developer to make a separate site for being able to conduct exams online. They just have to register on the site and enter the required information. Students can give exam without the need of going to any physical destination. They can view the result at the same time. Thus the purpose of the site is to provide a system that saves the efforts and time of both Educational organizations and the students.

- Responses by the candidates will be checked automatically and instantly.
- Online examination will reduce the hectic job of assessing the answers given by the candidates.
- Being an integrated Online Examination System it will reduce paper work.
- Can generate various reports almost instantly when and where required.

SCOPE OF THE PROJECT:

Scope of this project is very broad in terms of other manually taking exams. Few of them are :-
This project would be very useful for educational institutes where regular evaluation of students' is required. Further it can also be useful for anyone who requires feedback based on objective type responses.

Scope of this project is very broad in terms of other manually taking exams. Few of them are :-

- This can be used in educational institutions as well as in corporate world.
- Can be used anywhere any time as it is a web based application(user Location doesn't matter).
- No restriction that examiner has to be present when the candidate takes the test.

Features:

- Secure
- Easy to use
- Reliable and accurate
- No need of examiner

Chapter 2 LITERATURE REVIEW

Introduction:

Large language models (LLMs) have emerged as powerful tools in natural language processing (NLP), reshaping the landscape of AI applications. Among these, Transformer-based architectures, such as BERT, GPT, and their variants, have garnered significant attention for their ability to understand, generate, and interact with human language. This literature review explores the evolution of LLMs, the significance of Transformer architectures, and recent advancements in fine-tuning techniques, with a focus on Parameter-efficient Fine-tuning (PEFT) and its variants.

Large language models are advanced AI algorithms designed to understand, generate, and interact with human language. They are trained on vast datasets and can perform a range of language-related tasks. In recent years, these large language models have been developed primarily using Transformer-based architectures (Transformers)

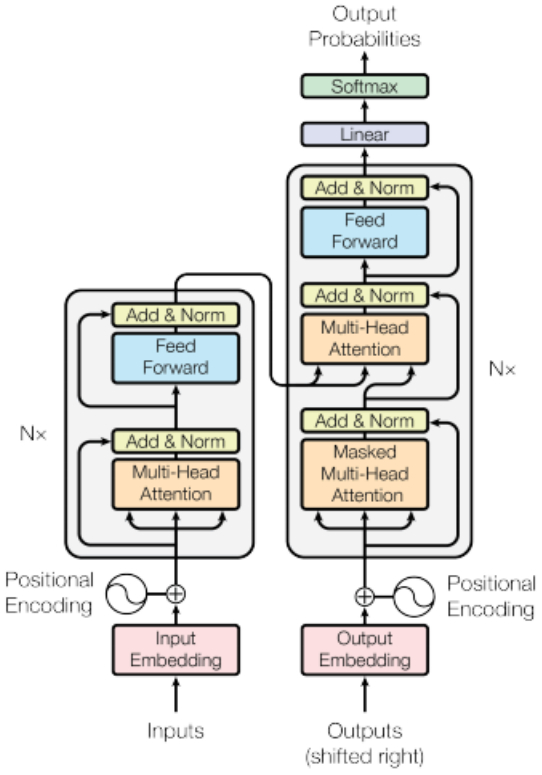


Figure 1: The Transformer - model architecture.

Transformers in NLP:

Transformers, introduced in the seminal paper "Attention Is All You Need" by Vaswani et al. (2017), have revolutionized NLP by overcoming limitations posed by earlier models like Recurrent Neural Networks (RNNs), GRUs, and LSTMs. Key reasons for their superiority include parallel processing capabilities, efficient handling of long-distance dependencies through self-attention mechanisms, and scalability to handle large datasets. The core architecture consists of an encoder and a decoder, both utilizing self-attention mechanisms.

Indeed, Transformers are revolutionizing the field of natural language processing (NLP), setting new benchmarks, and representing the technological frontier. They find utility in various applications including language translation, interactive chatbots, and enhancing search engine capabilities.

Today Transformers are a hot topic in Deep Learning, and this article describes their functioning and how have they surpassed the former champions of sequence-related challenges, such as Recurrent Neural Networks (RNNs), GRUs, and LSTMs. In short, we can summarize that the main reasons are as follows.

1. Parallel Processing: Transformers allow for parallel processing of input data, unlike RNNs which process data sequentially. This parallelism significantly speeds up training and makes easier handling large datasets.
2. Long-Distance Dependencies: Transformers, with their self-Attention mechanisms, are better at capturing long-distance dependencies in text. They can effectively link information across lengthy sequences, a task where RNNs often struggle due to their sequential nature.
3. Scalability: Transformers scale more effectively with the increase in model size and data. They can be efficiently trained on extensive datasets and can

4. be expanded to have many parameters, which is crucial for capturing the complexity of human language.

Transformer Architecture

Most foundation models use the transformer architecture. Let's look at the definition:

A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. It is used primarily in the fields of natural language processing and computer vision.

In 2017 transformers were introduced: Attention is all you need. They are the next generation of Recurrent Neural Networks and Long Short-Term Memory architectures and have several benefits:

- Parallel processing: Increases performance and scalability
- Bidirectionality: Allows understanding of ambiguous words and coreferences

The original transformer architecture defines two main parts, an encoder and a decoder. However, not all foundation models use both parts. BERT only uses encoders, GPT only decoders. More on this later.

Attention

Both encoders and decoders use the concept of 'attention'. Attention basically means to focus on the important pieces of information and to blend out the unimportant pieces. I like to compare this with 'fast reading'. Rather than reading full articles or even full books, I often browse chapter titles, first words of paragraphs and scan paragraphs for keywords to find what I'm looking for.

The words of an article, the parts of an image or the words in a sentence that should get most attention change dependent on what you are looking for. Let's look at a simple example sentence:

"Sarah went to a restaurant to meet her friend that night."

The following words should get attention for the following queries:

- What? -> ‘went’, ‘meet’
- Where? -> ‘a restaurant’
- Who? -> ‘Sarah’, ‘her friend’
- When? -> ‘that night’

The pretrained models can be extended and customized for different domains and specific tasks. Layers can sometimes be reused without modifications and more layers are added

To determine the attention of words (more exactly tokens) ‘queries’, ‘keys’ and ‘values’ are used by encoders and decoders in transformers. All of them are presented in vectors. Keys are found for certain queries if they are closest to the query vector. Keys are an encoded representation for values, in simple cases they can be the same.

There are different algorithms to implement the attention concept. I think an easy way to understand how this can work is to rank words high that are often used together in sentences. For example, ‘where’ and ‘restaurant’ have probably a closer relation than ‘restaurant’ and ‘faith’. So, for the query ‘where’ the word ‘restaurant’ gets more attention.

on top. If layers need to be modified, the new training is more expensive. The technique to customize these models is called Transfer Learning, since the same generic model can easily be transferred to other domains.

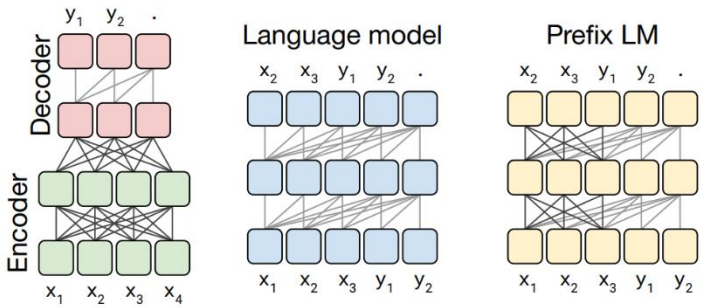


Figure 2: Type of Transformer architectures

Let’s explore the 3 basic transformer architectures in a simple, intuitive way.

1. Encoder Only Models:

uses the encoder part of the transformer architecture so that it understands semantic and syntactic language information. The output of BERT are embeddings, not predicted next words. To leverage these embeddings, other layer(s) need to be added on top, for example text classification or questions and answers.

BERT uses a genius trick for the training. For supervised training it is often expensive to get labeled data, sometimes it's impossible. The trick is to use masks as I described in my post Evolution of AI explained via a simple Sample. Let's take a simple example, an unlabeled sentence:

"Sarah went to a restaurant to meet her friend that night."

This is converted into:

- Text: "Sarah went to a restaurant to meet her MASK that night."
- Label: "Sarah went to a restaurant to meet her friend that night."

Note that this is a very simplified description only since there aren't 'real' labels in BERT.

In other words, BERT produces labeled data for originally unlabeled data. This technique is called Self-Supervised Learning. It works very well for huge amounts of data.

In masked language models like BERT, each masked word (token) prediction is conditioned on the rest of the tokens in the sentence. These are received in the encoder which is why you don't need a decoder.

Perfect for: Sentiment analysis, Named entity recognition, Sentiment classification

Popular models: BERT, ROBERTA

Popular apps: Google Search, in 2019, integrated BERT into its search algorithm to better understand the nuances and context of search queries.

Email services like Gmail, Outlook, and Yahoo also use BERT-type models to sort emails into spam, promotions, updates, etc

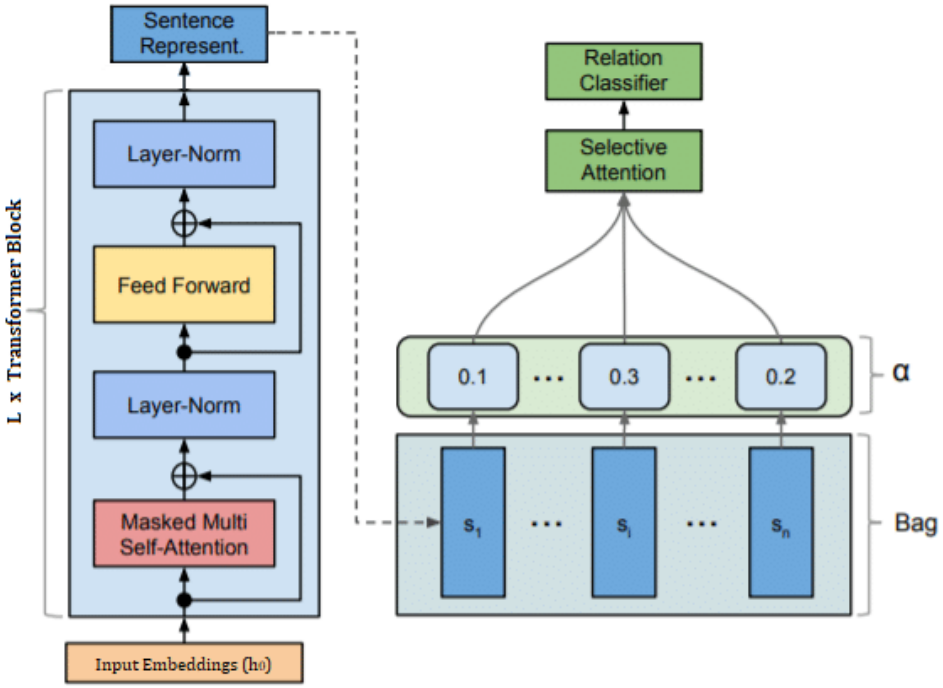


Figure 3: Encoder-only or Autoencoding models.

2. Decoder Only Models:

This is the most popular transformer architecture power LLMs like GPT and Llama. In language scenarios decoders are used to generate next words, for example when translating text or generating stories. The outputs are words with probabilities.

Decoders also use the attention concepts and even two times. First when training models, they use Masked Multi-Head Attention which means that only the first words of the target sentence are provided so that the model can learn without cheating. This mechanism is like the MASK concept from BERT.
After this the decoder uses Multi-Head Attention as it's also used in the encoder. Transformer based models that utilize encoders and decoders use a trick to be more efficient. The output of the encoders is feed as input to the decoders, more precisely the keys and values. Decoders can invoke queries to find the closest keys. This allows, for example, to understand the meaning of

the original sentence and translate it into other languages even if the number of resulting words and the order changes.

GPT doesn't use this trick though and only use a decoder. This is possible since these types of models have been trained with massive amounts of data (Large Language Model). The knowledge of encoders is encoded in billions of parameters (also called weights). The same knowledge exists in decoders when trained with enough data.

Perfect for: Text generation, summarization, code suggestion, and many more

Popular models: GPT-4, Llama2, Mistral-7B, Claude-V2 etc.

Popular apps: Of course, ChatGPT which is based on either GPT-4 or GPT-3.5,llama 2

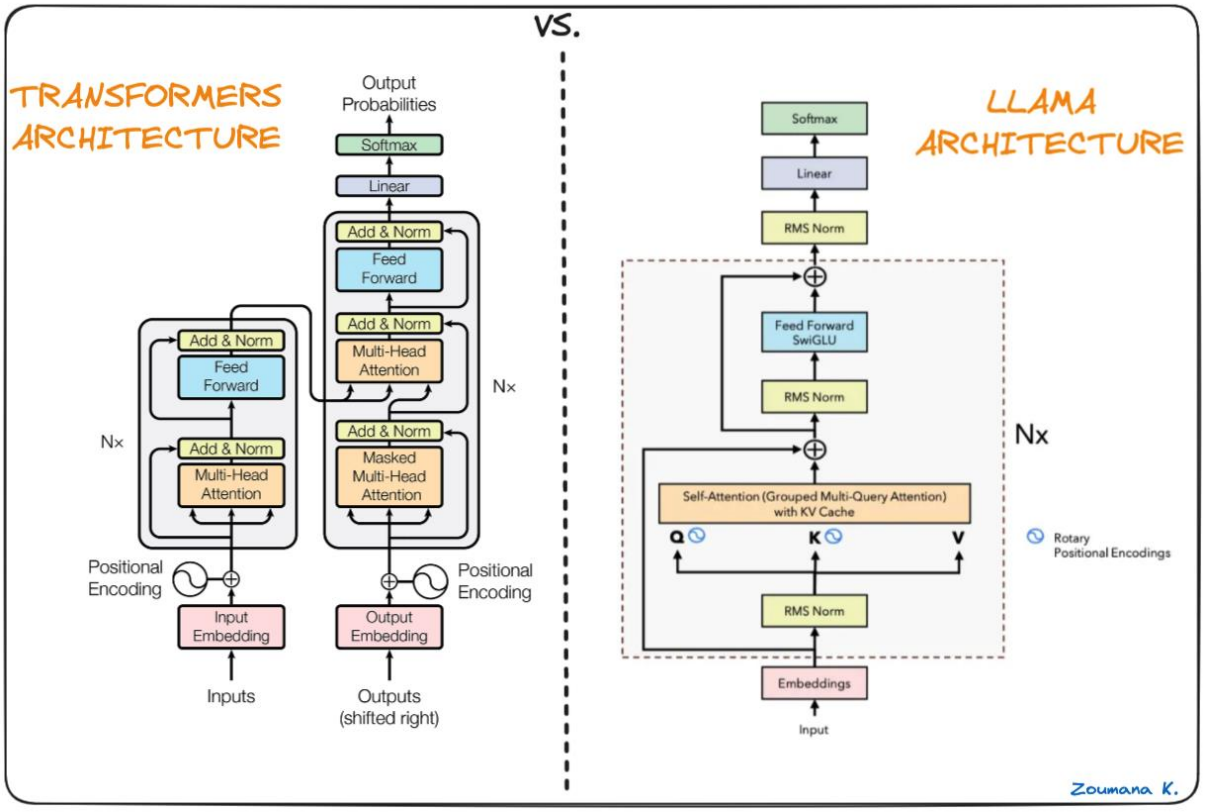


Figure 4: Comparison of Transformer and Llama Architectures for Language Modeling.

Encoder-Decoder Models

Imagine a skilled translator who not only understands the source language but can also accurately convey the essence in another language. The Encoder-decoder (aka Sequence-to-Sequence) models operate on this principle i.e.

understand first then generate. The encoder processes the input text and transforms it into a meaningful representation, while the decoder then generates an output based on this representation. Just like the encoder model, the encoder-decoder model is trained to predict purposefully hidden spans (similar to phrases) in a sentence.

Perfect for: Translation, text-to-speech, speech-to-text

Popular models: T5, BART

Popular apps: Google Translate is a famous example of an encoder-decoder model. OpenAI's Whisper automatic speech recognition AI also uses this architecture

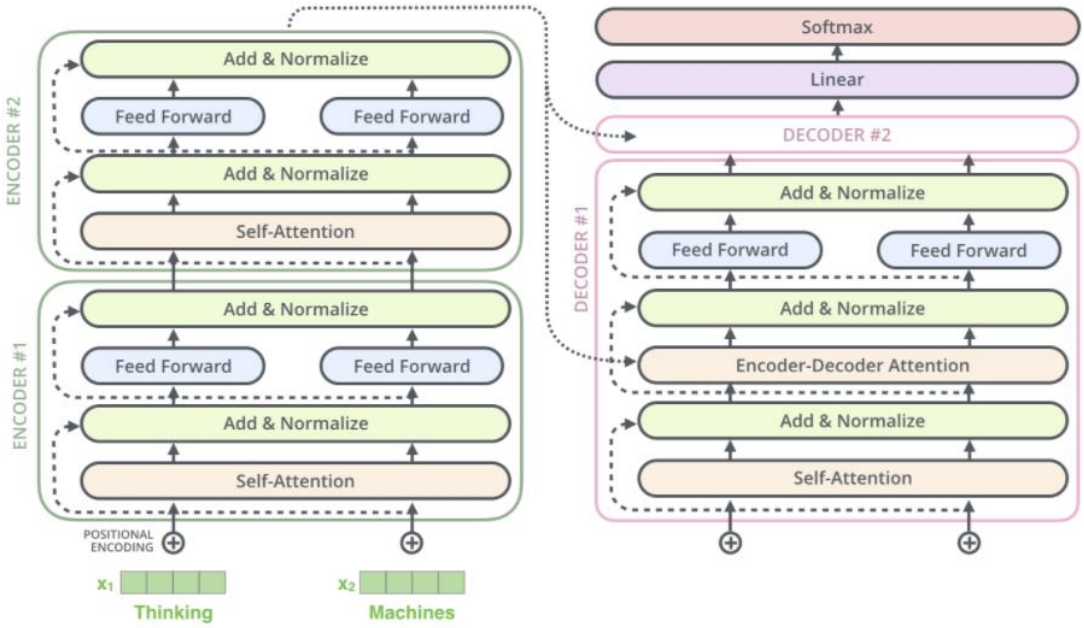


Figure 5: Encoder-decoder or Sequence-to-sequence models.

Fine tuning:

Fine tuning is a process where a pre-trained model is further trained on a custom dataset to adapt it for particular tasks or domains. In context of LLM's fine tuning involves training the model like Llama, Falcon and other open-source models on specific dataset to enhance its performance in specific contexts.

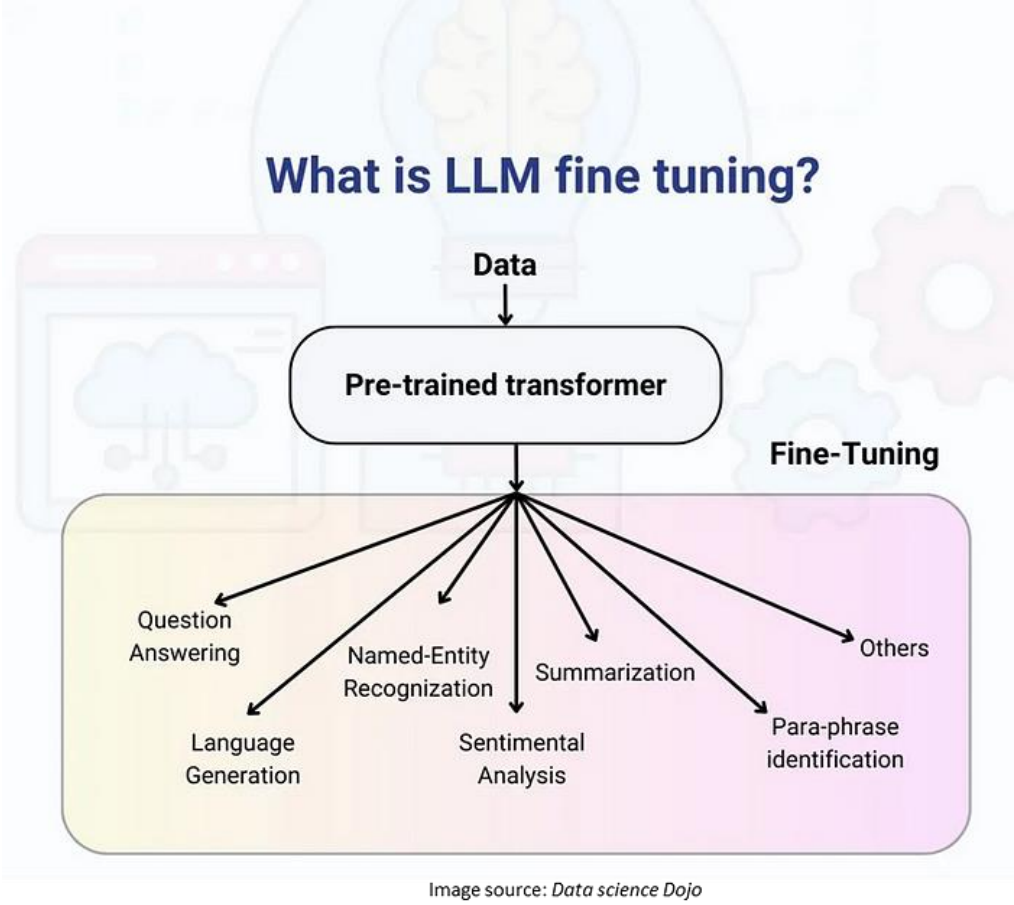


Figure 6: The process of LLM fine-tuning .

Why fine tuning?

We also should know why fine-tuning is so helpful and important in the field of Natural Language Processing. Even though pre-trained models are trained on large datasets and have a broad understanding of language, they might not be specialized in certain tasks. Fine-tuning adjusts the model to be more proficient in specific tasks like sentiment analysis, question answering, or text summarization, Customer Support, Medical Inquiries. Fine-tuning adapts the model to different domains (like legal, medical, or technical fields) specific tasks, improving its accuracy and effectiveness in those areas.

For example, fine-tuning a Large Language Model (LLM) like GPT for question answering, especially when it's already trained on the English language, can

significantly improve its performance and accuracy in this specific task. In order to fine-tune it for question answering for medical inquiries, the model should train on Q&A dataset and it learns understand the questions and how to retrieve and formulate appropriate answers.

Challenges of fine tuning:

Fine-tuning LLM's offers numerous benefits, but it also comes with significant challenges. Depending on the size of the model and the fine-tuning dataset, the process can take a significant amount of time and also high-performance GPUs or TPUs are often required to handle the computation load. LLM's are large in size and storing the parameters of these models, especially when multiple versions are maintained (pre-trained and fine-tuned models), requires considerable storage capacity. When an LLM is fine-tuned on a specific task or dataset, the model can perform better in that area, losing its ability to perform well on more general tasks it was originally trained on.

Parameter-efficient Fine-tuning (PEFT):

Parameter-efficient Fine-tuning overcomes the problems of consumer hardware, storage costs by fine tuning only a small subset of model's parameters significantly reducing the computational expenses while freezing the weights of original pretrained LLM. Additionally, this resolves the problem of catastrophic forgetting, which is a behavior seen when LLMs are fully adjusted.

When employing PEFT methods, the amount of storage required is only a few MBs for each downstream dataset, while still attaining performance comparable to full fine-tuning. For example, with full fine-tuning, 40GB of storage is required for each downstream dataset. The pretrained LLM is combined with the small, trained weights from PEFT techniques and this model is used for numerous tasks. This method of fine tuning helps to get

performance similar to full fine-tuning with less trainable parameters. There are several Parameter-efficient fine-tuning techniques and as follows:

- Adapter
- LoRA
- Prefix tuning
- Prompt tuning
- P-tuning

LoRA:

LoRA (Low-Rank Adaptation of Large Language Models) is a fine-tuning technique to train LLM's on specific tasks or domains. This technique introduces trainable rank decomposition matrices into each layer of transformer architecture and reduces trainable parameters for downstream task while keeping the pre-trained weights frozen. LoRA papers says that this method can minimize the number of trainable parameters by up to 10,000 times and the GPU memory necessity by 3 times while still performing on par or better than fine-tuning model quality on various tasks.

Adaptors fine tuning method has Inference latency problem which is resolved by LoRA. It adds values to transformers instead of adding layers. A large matrix is expressed as the product of two smaller matrix in low-rank decomposition. This assumes that redundant information is often easily stored in a big matrix, especially in high-dimensional spaces.

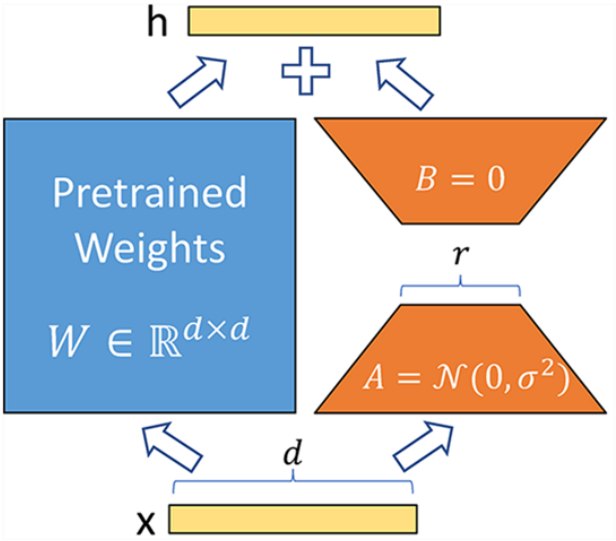


Figure 7: This diagram illustrates the LoRA fine-tuning technique.

Rather than altering the weight matrix W of a layer in all of its components, LoRA creates two smaller matrices, A and B, whose product roughly represents the modifications to W. The adaptation can be expressed mathematically as Y = W+AB, where A and B are the low-rank matrices. If W is an mxn matrix A might be mxr and B is rxn where r is rank and much smaller than m,n. During fine tuning only A and B are adjusted enabling the model to learn task specific features.

QLoRA Overview: QLoRA is an advanced version of Low-Rank Adaptation (LoRA) that reduces the precision of weight parameters in pre-trained large language models (LLMs) to 4-bit, significantly decreasing memory requirements. Typically, models store parameters in a 32-bit format, but QLoRA compresses them to 4-bit, allowing fine-tuning on a single GPU and enabling LLM deployment on less powerful hardware.

Key Innovations:

1. **4-bit NormalFloat (NF4):** An optimal quantization data type for normally distributed data, outperforming 4-bit Integers and Floats.

2. **Double Quantization:** Quantizes quantization constants, saving about 0.37 bits per parameter, which reduces memory significantly for large models.

3. **Paged Optimizers:** Utilizes NVIDIA unified memory to manage memory spikes during processing, facilitating efficient data transfers between CPU and GPU.
4. **Memory Efficiency and Performance:** QLoRA compresses models using NF4, decreasing memory usage while maintaining or even improving performance compared to LoRA and base models. However, training time increases due to the quantization and dequantization processes.
5. **4-bit NormalFloat:** NF4 is designed for AI applications to efficiently quantize neural network weights, crucial for deploying large models on less powerful hardware. It accurately represents weights within its bit constraint, unlike general-purpose 4-bit Floats, which have limited precision and range.
6. **Quantization and Double Quantization:** Quantization reduces model size by converting high-precision data to lower precision, e.g., FP32 to 4-bit NF4. Double quantization further reduces memory by quantizing the constants themselves, crucial for large LLMs.
7. **Paged Optimizers:** Paged optimizers handle memory usage during large model training by transferring data between CPU and GPU as needed, using NVIDIA unified memory to manage memory spikes effectively.

Vision Language Models (VLMs)

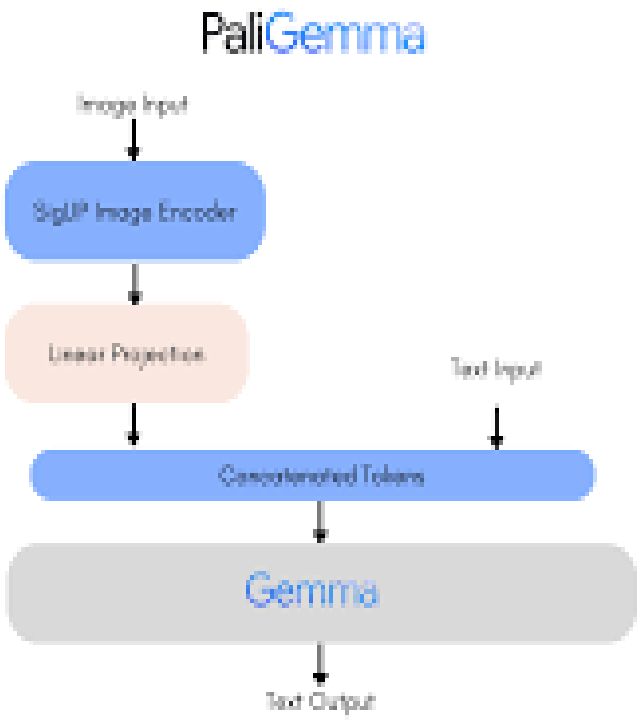


Fig 8 : Vision Language Models (VLMs)

Vision Language Models (VLMs) bridge the gap between visual and linguistic understanding of AI. They consist of a multimodal architecture that learns to associate information from image and text modalities. In simple terms, a VLM can understand images and text jointly and relate them together. By using advances in Transformers and pretraining strategies, VLMs unlock the potential of vision language applications ranging from image search to image captioning, generative tasks, and more!

Chapter 3 Dataset

Initially, our team explored leveraging semantic similarity models like SBERT and ST5. These models excel at comparing two sentences and generating a similarity score, widely used in applications like retrieval augmentation generation. However, during discussions, we identified limitations by relying solely on semantic similarity for our automatic essay grading model.

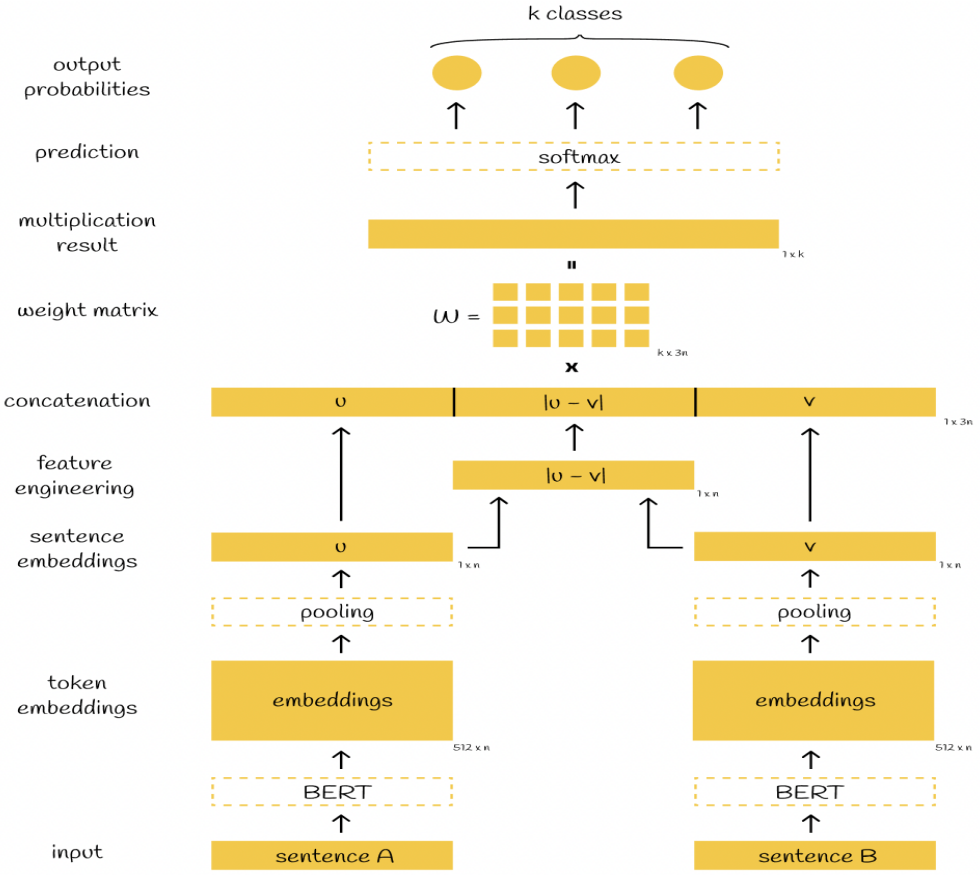


Figure 9:siamses system

This Score is useful in many applications like retravel augmentation generation.

But after some argue we reach that siamese similarity not ideal form for our model to build on because many reasons:

1. modern LLMs, primarily built on decoder-only architectures, lack the encoder component crucial for Siamese networks employed by semantic similarity models.
2. The binary true/false format for comparing a student answer to a reference answer doesn't fully utilize the potential of our vast 7B-parameter LLM.

capitalize on the LLM's full capabilities, we adopted a different approach. We believe the optimal scenario involves providing the model with the question, student answer, and essay context (three elements) and training it to predict the "truthfulness" of the answer as True or False. This methodology allows the LLM to leverage its internal knowledge and reasoning abilities to make informed judgments, exceeding the limitations of simple pairwise comparisons.

Example:

Instruction: check answer is true or false of next question using context below:

Read this: The fourth a cappella musical to appear Off-Broadway, In Transit, premiered 5 October 2010 and was produced by Primary Stages with book, music, and lyrics by Kristen Anderson-Lopez, James-Allen Ford, Russ Kaplan, and Sara Wordsworth. Set primarily in the New York City subway system, its score features an eclectic mix of musical genres (including jazz, hip hop, Latin, rock, and country).What is the name of the a cappella musical that debuted Off-Broadway in the same year as Perfect Harmony?

#Student answer: The name of the a cappella musical that debuted Off-Broadway in 2010 is "In Transit.

#Response:

Instruction: check answer is true or false of next question using context below:

تقنية النانو أحد االساليب المبتكرة لدراسة المادة وطرق تغييرها عند مستوى النانو؛ من أجل إنتاج

مواد أخرى متطورة تخدم
البشرية فى مختلف مجاالت الحياة، والنانو وحدة قياس دقيقة جدا ؛ فالنانو الواحد
يعادل واحدا على المليون من المليمتر ؛ لذلك تستحيل رؤية ألشياء المقاسة بالنانو بواسطة العين المجردة،
أو حتى بمكبرات الرؤية البدائية، وهى تستخدم فى القياس الذرى لتحديد ألحجام الخاصة بجزئيات المادة
المتواجدة بها.
#Student answer: لأنها كبيرة جدا
#Response: Model response will output :خطأ

After response model will predict True or False

Ok that is all of course no!!!

Our dataset:

There is no dataset on whole internet look like this no dataset contain context question answer (this answer be True or False) and label (is True or False) .

Approach making dataset:

so we can make it three approach we can use:

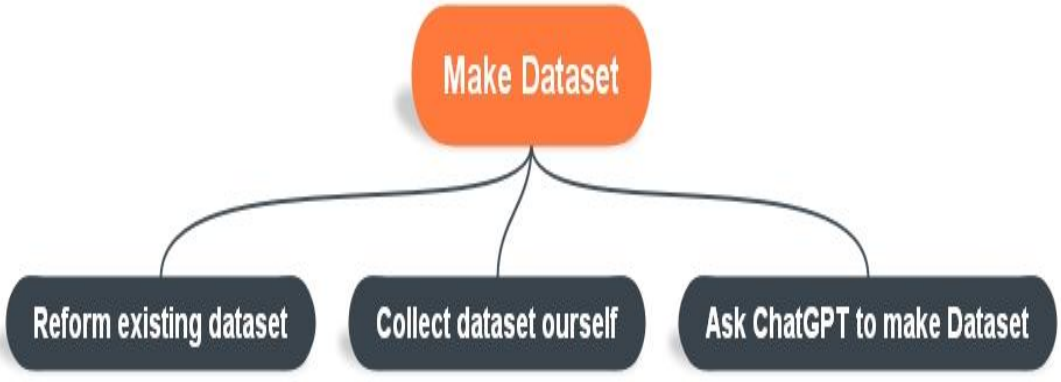


Figure 10 Different approaches to make dataset

Ask ChatGPT:

Models like GPT4 and ChatGPT are usually trained on huge datasets to learn multiple knowledge bases and can-do various tasks. We can use this property of LLMs to generate datasets which are more specific to our needs.

Now, we can generate synthetic datasets from scratch, it is always better to provide the model with some examples to begin with.

Let us discuss the EXACT pipeline we use to generate high-quality synthetic Datasets.

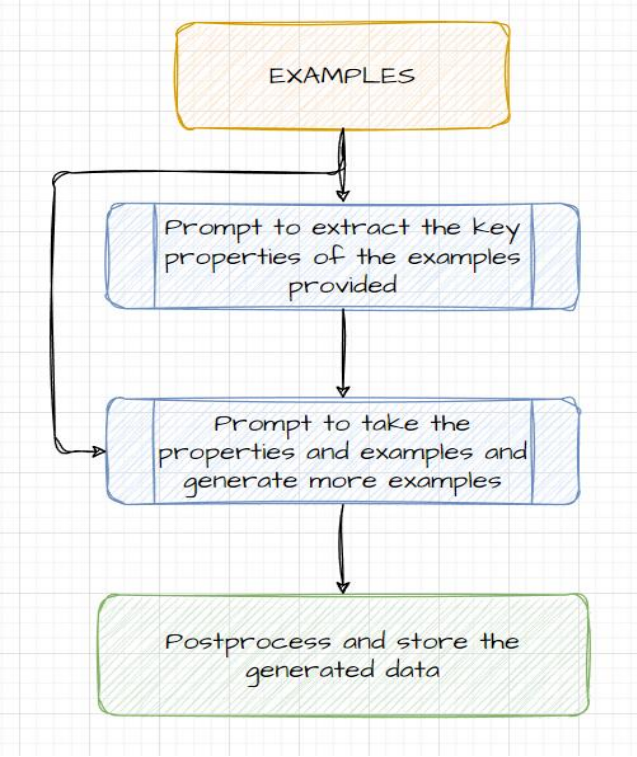


Figure 11: diagram shows the steps used to make dataset using ChatGPT

- Extracting properties from provided examples

The first step is to analyze the examples provided and understand what exactly we are seeking to achieve. This is not required, but we have found that this increases the quality of generated outputs.

The goal of this step is to generate and understand the key properties of the provided examples. In the dataset, pairs of inputs and outputs are generated, these properties are the guide for the model to generate them.

- Generating the dataset

Once we have the properties, we want our dataset to have, we can begin to generate the synthetic dataset we want. We can tweak the properties if we want, this gives us more control over the generated dataset.

For this, we usually set the model parameters in a specific way:

- **Temperature:** The temperature is usually set to the max value, 1. This is because we want the model to be as "random" as possible, while still staying within the constraints provided, to generate the most diverse dataset possible.
- **Frequency Penalty:** This value is also set to a higher number, 1. A higher frequency penalty value will encourage the model to produce more diverse and unique content by penalizing the repetition of the same words or phrases.
- **Presence Penalty:** Setting this parameter is a bit tricky. Higher presence penalty that model will be penalized if it generates the same word multiple times. This can be good if you want your dataset to be diverse and contain lots of different words. But if you are generating something extremely specific, you might want to have the same words present in your dataset multiple times in different contexts. The value of this parameter depends entirely on your requirement.

You can of course tweak these parameters according to your needs.

This efficient method is a fast way to make dataset which is used in alpaca model like we want but unfortunately excessive cost $**600 dollar to make just 52k sample.**

collect Dataset:

a. Establish Data Collection Channels:
   - Collaborate with educators or institutions to gather student essays (ensuring ethical permissions and data privacy).
   - Conduct online essay contests or surveys with incentives for participation.
   - Utilize crowdsourcing platforms for essay collection and annotation.

b. Develop Data Collection Platform: If collecting directly, create a secure and user-friendly platform for essay submission and data management.

c. Implement Anonymization Protocols: Remove personally identifiable information to protect student privacy.

d. Review and Annotate: Assess essays for quality, relevance, and alignment with project goals, annotating them with scores or feedback.

e. Ensure Diversity: Strive for a dataset representing diverse topics, writing styles, and student backgrounds.

Figure 12 : A schematic illustrating various data collection methods.

These methods are grouped into ten categories: forms and questionnaires, interviews, observation, documents and records, focus groups, oral histories, combination research, online tracking, online marketing analytics, and social media monitoring.Very time-consuming way to make dataset and need larger team to make (number of team 6) Access and quantity limitations

Reform existing dataset:

While reforming existing datasets can be a valuable approach to building your training data, it requires careful consideration due to its inherent complexity. Here is how we tackled this challenge:

1. Identifying Relevant Datasets:

We explored online repositories like Kaggle, Hugging Face Datasets, and Project Gutenberg for essay datasets with the specific format of "context-question-answer." There two datasets have specific form:

- Squad
- QuAC

What is QuAC?

Question Answering in Context is a dataset for modeling, understanding, and participating in information seeking dialog. Data instances consist of an

interactive dialog between two crowd workers: (1) a student who poses a sequence of freeform questions to learn as much as possible about a hidden Wikipedia text, and (2) a teacher who answers the questions by providing short excerpts (spans) from the text. QuAC introduces challenges not found in existing machine comprehension datasets: its questions are often more open-ended, unanswerable, or only meaningful within the dialog context.
QuAC is meant to be an academic resource and has significant limitations. Please read our detailed datasheet before considering it for any practical application.

Is QuAC exactly like SQuAD 2.0?

No, QuAC shares many principles with SQuAD 2.0 such as span based evaluation and unanswerable questions (including website design principles! Big thanks for sharing the code!) but incorporates a new dialog component. We expect models can be easily evaluated on both resources and have tried to make our evaluation protocol as similar as possible to their own.

Two dataset have same structures:

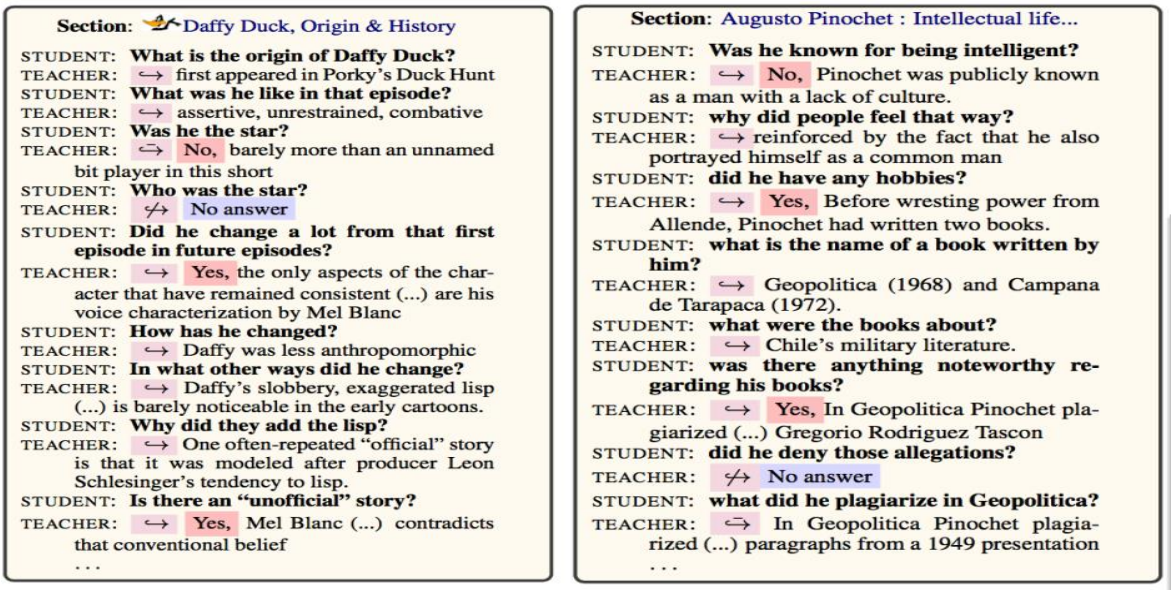


Figure 13:Quac dataset

*Here section is representing context

We selected the Quac dataset based on its answer length, which closely resembles the type of responses students provide in essay questions
Now how we go from context-question-answer format to our desire format
Format Conversion: To bridge the gap between Quac's format and our desired format, we identified answers labeled as "True" as the ground truth outputs for our model, representing the ideal responses to each context-question pair. Generating "False" answers,

Second how we identify False answers:

- Choose random answers as False answers

We try this method firstly try it on model flan t5, Scheduler : constant, learning rate=1e-3, Optmizer: adafactor it is model couldn't difference between answers However, it presented an initial hurdle. Randomly selecting alternative options with our flan T5 model yielded underwhelming results, with an accuracy of only 40%.

- Using **RAG (Retrieval Augmented Generation)** to choose similar answer to our true answer

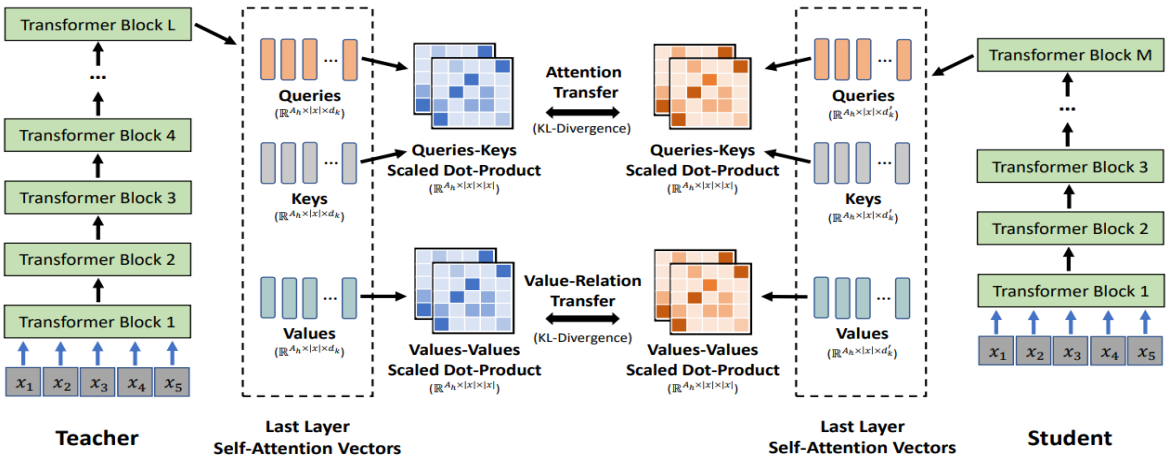


Figure 14: This diagram shows how a RAG (Retrieval Augmented Generation) model work

Seeking to enhance the model's learning challenge and ability to discern subtle differences, we implemented RAG (Retrieval Augmented Generation). RAG enabled us to identify answers most like the genuine "True" answers for each context-question pair. By setting the "False" answer as the third most similar option, we created a more nuanced distinction for the model to learn from, driving its ability to recognize subtle variations.

Boosted Accuracy with Fine-tuned RAG: This strategic approach, coupled with leveraging an all-MiniLM-L6-v2 model for RAG, resulted in a significant accuracy jump to 88%. This outcome highlights the effectiveness of utilizing RAG and carefully crafting "False" answers to develop a robust training dataset for our essay grading model.

This is whole steps for making a dataset to our model, but we remain after all this need to crucial step which is augmentation answers to make model learn more.

This is very efficient way to make dataset to our model we find that their better dataset than Quac Quac is particularly good dataset, but we need the

best, so we go to open orca **dataset, what** is open orca dataset?

The Open Orca Dataset is structured in a tabular format with data instances, fields, and splits.

Each data instance represents entries from the FLAN collection, which have been augmented by submitting the listed question to either GPT-4 or GPT-3.5. The response from the AI is then entered into the response field.

Data fields include 'id', a unique identifier; 'system prompt', representing the System Prompt presented to the GPT-3.5 or GPT-4 API for the datapoint; 'question', a question entry as provided by the FLAN Collection; and 'response', a response to that question received from a query to either GPT-3.5 or GPT-4.

The data split is currently at 17.6% test, providing a balance of data for training and evaluation purposes.

Summarize: it contains instructions promote for train model likes ChatGPT:

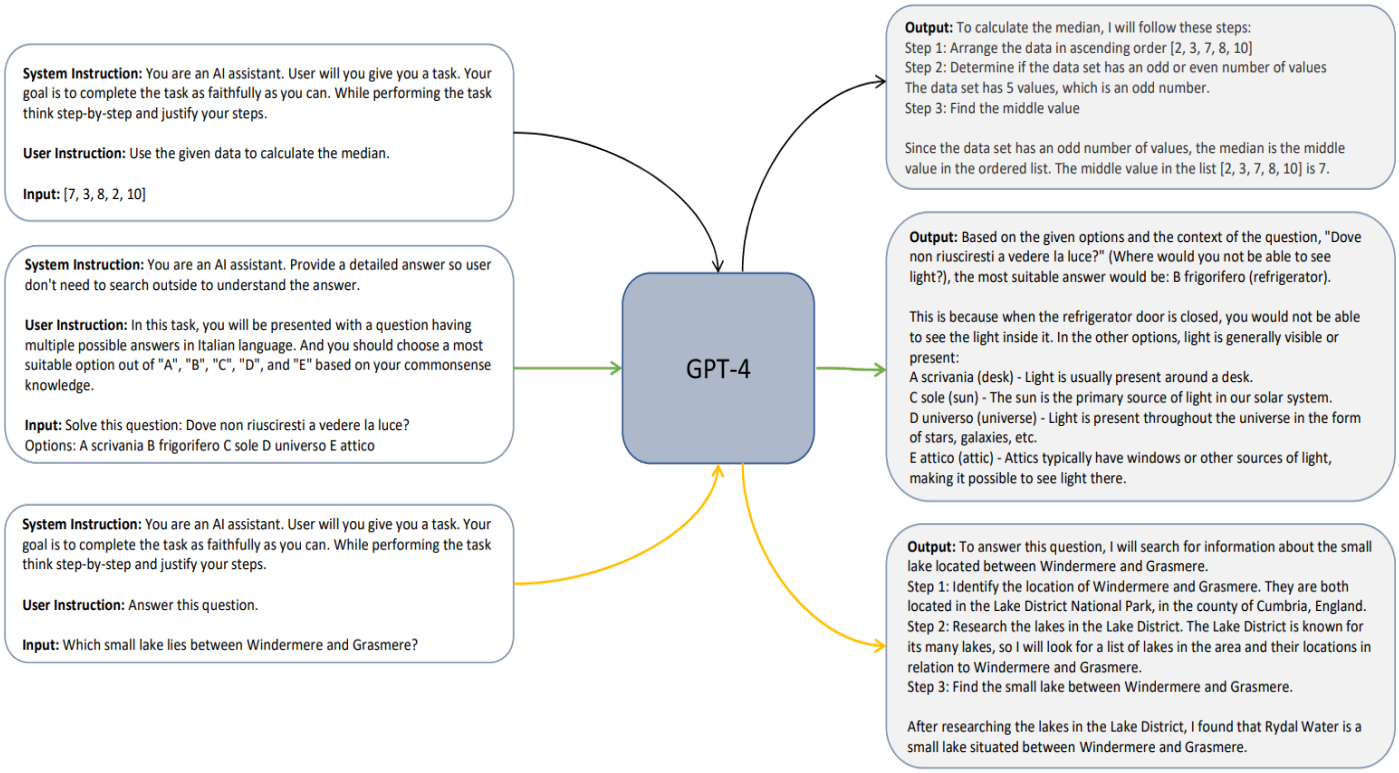


Figure 15 openorca structure

While establishing a robust dataset creation process, we recognized the pivotal role of data augmentation in maximizing model learning. Through a rigorous exploration of alternative datasets, Open Orca emerged as a superior candidate for fulfilling our specific requirements.

Open Orca: Unveiling a Valuable Resource for Model Advancement

- Structure: Open Orca presents a meticulously organized tabular format, encompassing instances, fields, and splits. Its entries are derived from the FLAN collection, augmented with responses generated by GPT-4 or GPT-3.5. Key data fields include unique identifiers, system prompts, questions, and responses. A 17.6% test split ensures balanced evaluation.
- Advantages over Quac: Open Orca distinguishes itself with the inclusion of instructional prompts, enabling the training of ChatGPT-like models.

Notably, it has demonstrated significant accuracy gains in LLMs, as visualized in the accompanying graph.

To understand how important instruction tune is look at below graph show difference in accuracy gpt before and after train on instruction tune

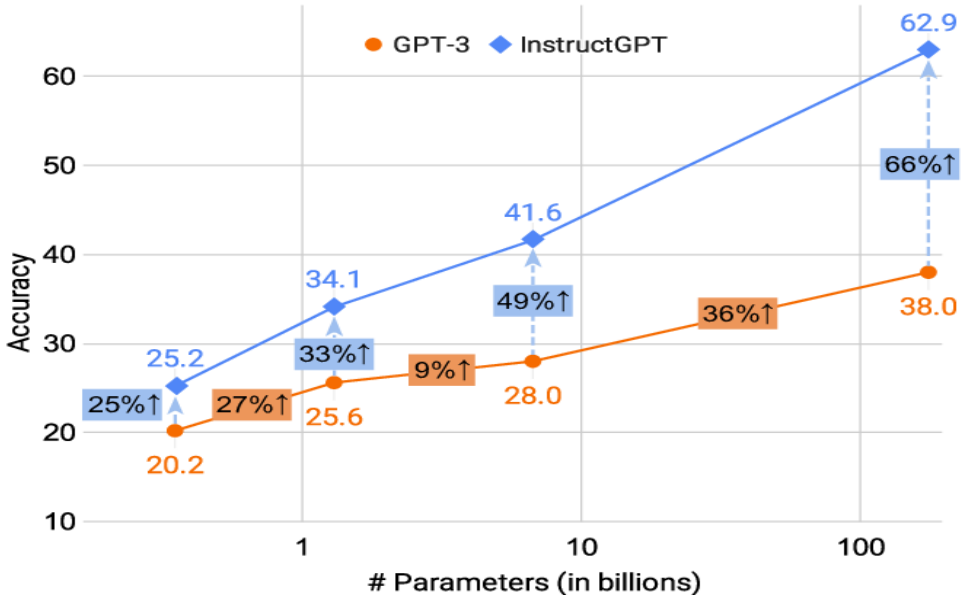


Figure 16 : improvement of instruction tuning on gpt

Strategic Data Extraction for Contextual Relevance

- Rigorous Filtering: To ensure the capture of contextually rich information, we implemented a multi-stage filtering process, meticulously examining over 1,000 initial prompts, and retaining only those containing contextual elements. These filtered prompts were subsequently employed to filter the 2.1 million samples, guaranteeing contextual alignment.

Dataset Creation: Adhering to Established Standards

- Replicating Proven Processes: We adhered to the identical dataset creation process utilized for the QuAC dataset, encompassing:

- Ground truth argument labeling
- Strategic selection of the third most correct answer to foster diversity
- Incorporation of incorrect answers for contrast
- Organization of the dataset with context, questions, student answers, and labels
- Shuffling to mitigate potential biases during training

Results: Attaining Remarkable Accuracy

- Elevated Performance: The English LLM achieved a remarkable 97% accuracy on evaluation metrics, while the Arabic LLM attained a commendable 90% accuracy.

There is last thing to make dataset perfect that is label, label is very important thing to improve accuracy that only make model predict True for right answer and False for wrong answer is bad way to use llms so we get idea to make model understand more context and task which do this by make llm predict the answer True or False + Ground True answer this method will make model understand context and produce more logical answers example:

صحيح أن الجواب هو الفقرة تنتمي إلى الفئة: المكان الطبيعي.

True the answer is Loki is ultimately imprisoned on S.H.I.E.L.D.'s flying aircraft carrier

Figure 1 : comparison between 2model after edit dataset .

Model	Accuracy before right answer	after
Flan t5	20 %	83 %
Bloomz	40 %	85 %

This accuracy compared before adding right answer to label make 60 % difference in accuracy so we can say it bratty work

Our robust English dataset laid the foundation, but the call for an Arabic counterpart resonated strongly. To empower truly multilingual LLMs, we embarked on a strategic translation journey.

Harnessing Google Translates Advancements:

- Instead of reinventing the wheel, we leveraged Google Translate's impressive progress. Citing "A Pragmatic Assessment of Google Translate for Emergency Department Instruction," which reports an 82.5% accuracy, we saw a promising foundation.

Beyond Accuracy: Nuanced Augmentation:

- While accuracy is crucial, we recognized the importance of context and domain relevance. Therefore, we implemented a multi-pronged approach:
    - Targeted Dataset Augmentation: Strategic techniques refine translations and optimize accuracy within our specific domain.
    - Human Expertise Integration: To ensure contextual fidelity and domain relevance, we involve the invaluable insights of human experts.

Building Robustness through Randomization:

- To equip the LLM for real-world scenarios, we incorporated a unique strategy: injecting random answers like single characters and meaningless words ("asdasjkldamsj","112ssa",'s'). This serves a dual purpose:

- Prevents Student System Hacking: Our experiments revealed the LLM's adeptness at mimicking long sentences. Random insertions discourage attempts to exploit this by feeding predictable patterns.
- Enhances False Answer Classification: By learning to identify and classify these nonsensical answers as "False," the LLM strengthens its overall classification accuracy.

This meticulous approach to building our Arabic dataset, coupled with the robust English counterpart, paves the way for powerful multilingual LLMs. These models will transcend language barriers, fostering knowledge exchange and understanding across diverse cultures and perspectives

Create Vision dataset:

We decide to train VLMs we need to give it image this image will contain two merged images as desgin below :

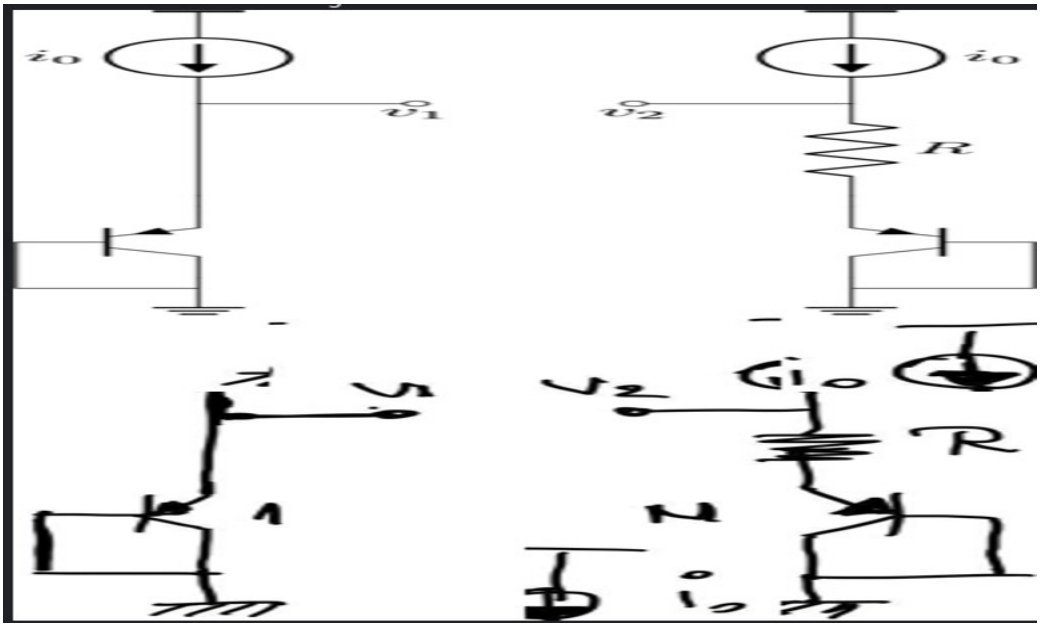


Figure 17 : Sample of image that we trained VLMs on it .

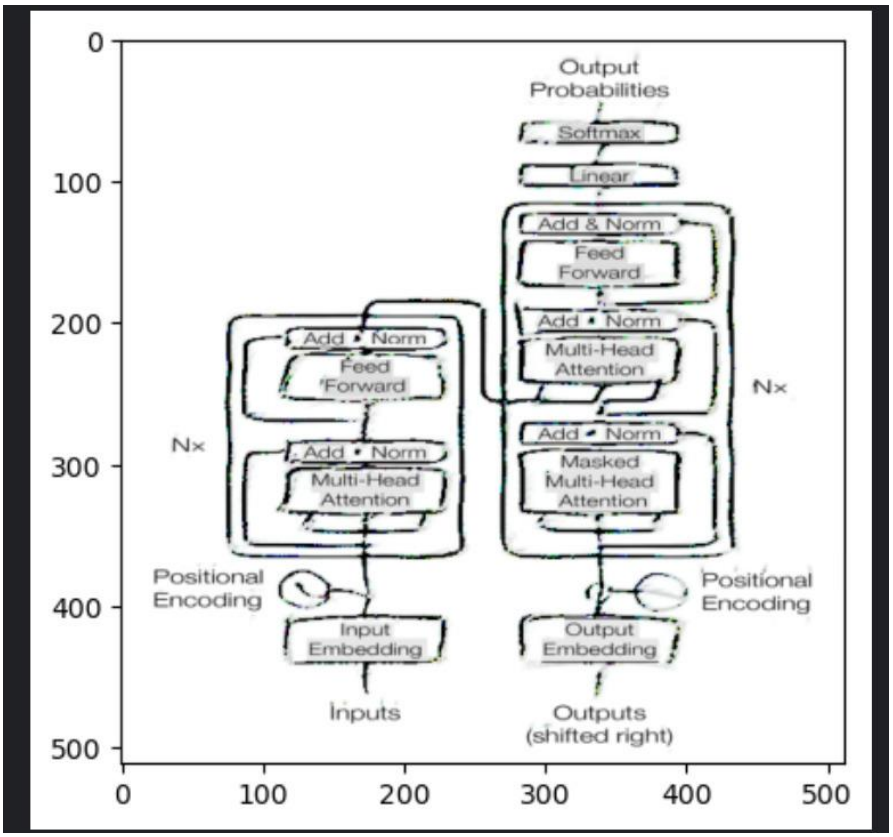
To maximize understanding and comparing between images we decide to put two images in one rather than two .

We need True human scratch and False human scratch.

There a dataset call paper2fig100k this conatain 100k image and captionion from arxiv papers so how make human scratch from it ?

By using Gans we able to make this by gave it to gans and ask it to generate image look like human draw scratch

It create acceptable imgaes .



But the GANs always corrupt text like below how we can solve this problem?

We use ocr this to extract images before generation and masked it before pass it to GANs

Then we repute it in again in image after images generated.

By this way we create True answers.

False answers by augmented the scratch images by three techniques:

1-replace high intensity regions(60%)

2-random sample (15%)

3-add random shape and lines to images (25%)

Then merged images with each pairs pith each other .

Chapter 4 Methodology

Exploring Transformer Architectures for Our LLM:

We delve into the realm of transformer architectures, evaluating three distinct approaches: encoder-only , encoder-decoder (T5), and decoder-only (GPT).

Moving Beyond Decoder-Only Hype:

A common misconception favors decoder-only models as the pinnacle of architecture. However, research in "Do We Still Need Clinical Language Models?" challenges this notion, demonstrating that even smaller models can outperform them. Such findings highlight the importance of moving beyond architectural trends and conducting thorough evaluations.

Our Three-Pronged Approach:

Guided by this principle, we experiment with three transformer architectures:

- BERT: This encoder-only model excels at understanding context and relationships within text, making it ideal for tasks like information retrieval and question answering.
- T5: The encoder-decoder architecture of T5 shines in tasks requiring generation and translation, allowing it to effectively process and respond to prompts and instructions.
- GPT: While decoder-only models like GPT might appear appealing due to their generative capabilities, we aim to objectively assess their suitability for our specific needs and tasks.
- By examining these diverse architectures, we aim to identify the optimal model configuration for our LLM, ensuring not only high performance but also suitability for the intended tasks and real-world application.

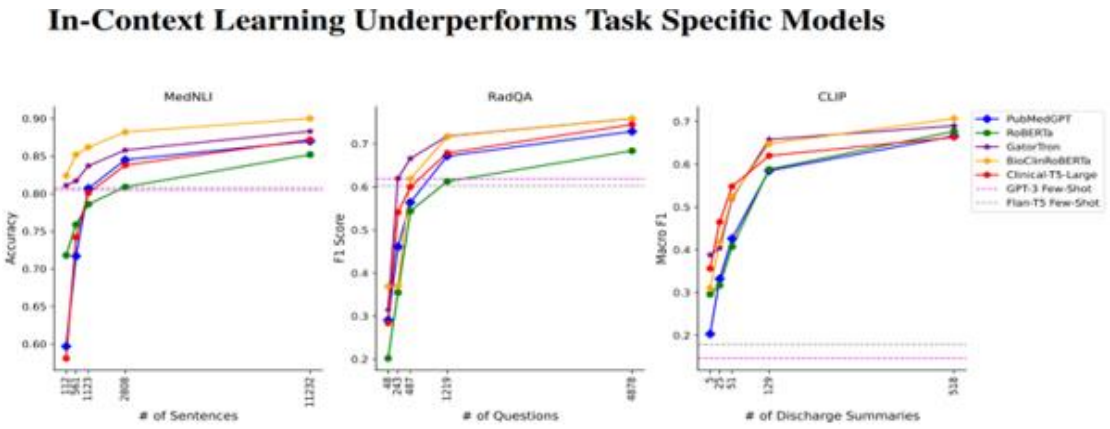


Figure 18 : accuracy of fine tuned model to gpt in context learning

Our Model Results:

Let's go to the model according to leadboard of hugging face which contain all fine tune model and state of art models

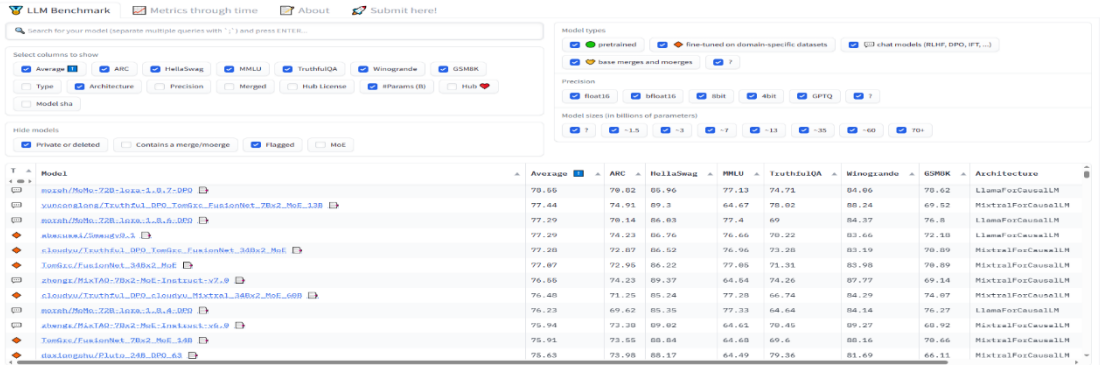


Figure 19 : hugging face lead board

All of it decoder only model We use 7b due to gpu capacity run 50h for each llm on gpu P100 on kaggle

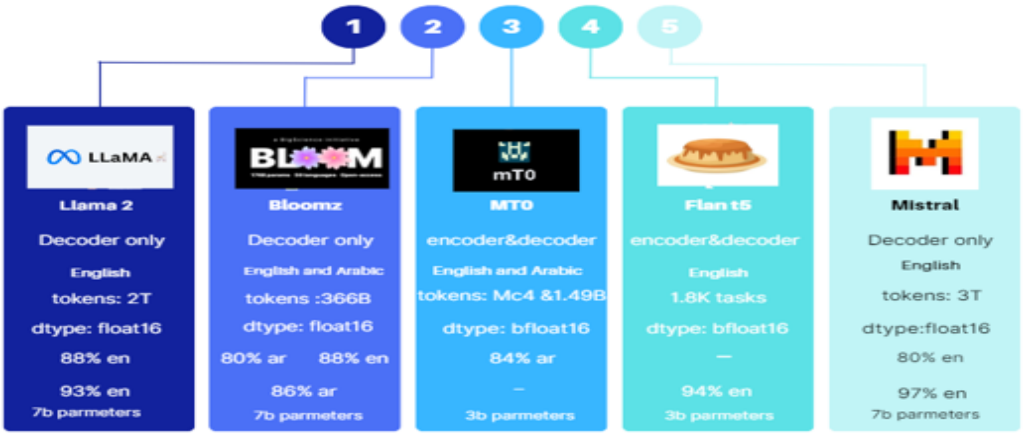


Figure 20 :models description and accuracy

This are models we use let's show accuracy

Results

Table 2: proposed model

model	Accuracy on Quac	Accuracy on openorca	epoch	lang	Architecture	size
Flan t5	90%	92%	3	en	Encoder decoder	3b
bloomz	93 %	_	4	ar	Decoder only	7b
roberta	_	_		en	Encoder only	
Marcoroni-7b-DPO-Mergea	**96%**	**97%**	2	en	Decoder only	7b
Llama 2	92%	92%	3	en	Decoder only	7b
MT0	_	85%	2	ar	Encoder decoder	3b
Mistral (orginal)	78 %	_				
MistralTrix-v1	91	_	2	en	Decoder only	9b
Bart	72%	_	1		Encoder decoder	700M

In out opinion trick of add right answer to label make it possible to train llms on classification problem.

We also try Siamese system using variety of models on paws dataset

	BERT	T5-large	BART
accuracy	84.5875%	83.9875%	82.5875%

Table 3 : Siamese system accuracy with models PAWS dataset

F1 score for Marcoroni-7b-DPO-Mergea and bloomz

	Bloomz	Marcoroni-7b-DPO-Mergea
accuracy	85.5875%	0.96%

Table 4 : F1 score for Marcoroni-7b-DPO-Mergea and bloomz

Let's talk about out technique and method use in our project:

First let’s identify tools like hugging face

Models:

Hugging face :

Hugging Face is a French-American company and a vibrant open-source community dedicated to democratizing Artificial Intelligence (AI) by providing powerful tools and resources for building, training, and deploying machine learning models, particularly focusing on Natural Language Processing (NLP) and Computer Vision (CV).

We use all models from it

Let's go on every model and say what we use on it

Siamese neural network Model:

A Siamese neural network is a type of deep learning model that can learn to measure the similarity or dissimilarity between two inputs. It consists of two identical subnetworks that share the same weights and parameters, and a distance function that computes the distance between the outputs of the subnetworks.

Siamese neural networks can be used for various tasks, such as face recognition, signature verification, or natural language processing. In this talk, I will present my Siamese neural network model that predicts if two sentences are similar or not, based on their semantic and syntactic features.

The model:

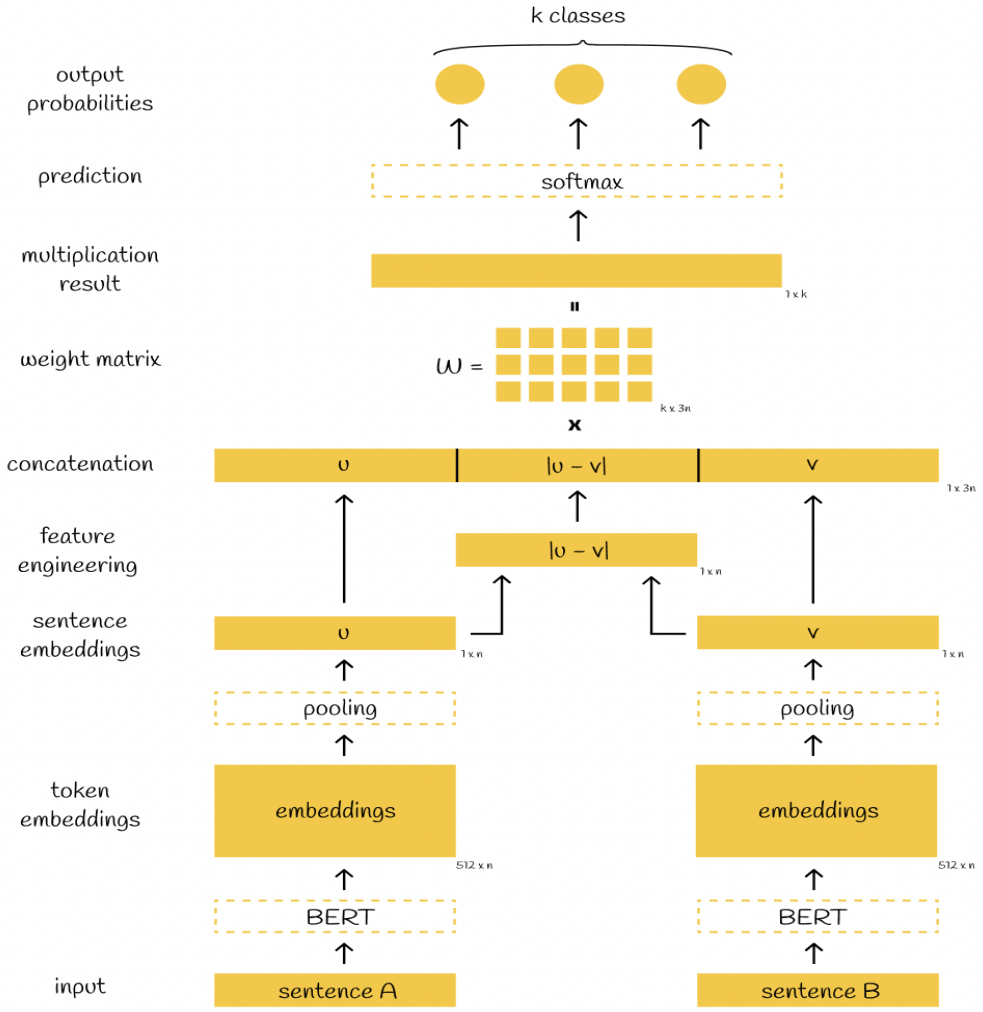


Figure 21 : Siamese neural networks

How it works:

1. **Input Sentences**:

   i. The diagram starts with two input sentences labeled "sentence A" and "sentence B."
2. **BERT Token Embeddings**:
   i. Both sentences are processed by BERT (Bidirectional Encoder Representations from Transformers) to obtain token embeddings.
   ii. Token embeddings capture contextual information for each word in the sentences.
3. **Sentence Embeddings**:
   i. The token embeddings are then pooled using mean pooling to get sentence embeddings for both sentences.
   ii. Sentence embeddings represent the overall meaning of the entire sentence.
4. **Feature Engineering**:
   i. Feature engineering involves concatenating the sentence embeddings and obtaining the absolute difference between them.
   ii. These features are used for downstream tasks like classification.
5. **Weight Matrix and Multiplication**:
   i. The features are multiplied by a weight matrix (denoted as W).
   ii. The multiplication result is passed through a softmax function to obtain output probabilities.
6. **Classification and Prediction**:
   i. The output probabilities correspond to true or false classes.
   ii. Predictions are made based on these probabilities.

Pooling:

Pooling is a technique commonly used in neural networks, particularly in deep learning models for natural language processing (NLP) and computer vision tasks. The purpose of pooling is to reduce the dimensionality of feature representations while retaining essential information.

1. CLS token

One common way of doing this is to use a special token, called the CLS token, that is prepended to the input sequence and learns to encode the global meaning of the sentence or paragraph.

The final hidden state of the CLS token can then be used as a representation for downstream tasks, such as classification or regression.

We can use CLS token in BERT directly but some encoders doesn't have CLS token like the T5 encoder.

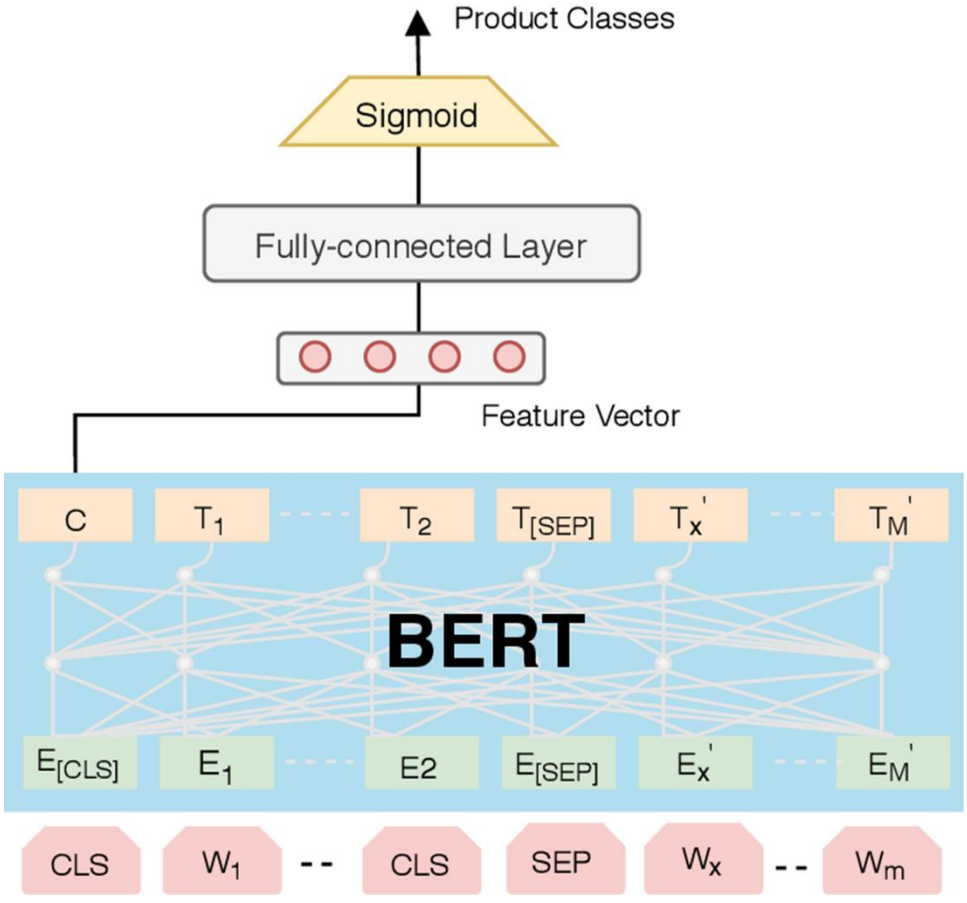


Figure 22 : BERT Architecture

2. Pooling last hidden states

Another way of pooling is to apply a function, such as mean, max, or attention, over the last hidden states of all the tokens in the input sequence. This produces a single vector that summarizes the whole input. For example, mean pooling

computes the element-wise average of the hidden states, while max pooling selects the highest value for each element. Attention pooling assigns a weight to each hidden state based on its relevance to the input and then computes a weighted sum. Pooling last hidden states can be useful for tasks that require a holistic understanding of the input, such as sentiment analysis or text summarization.

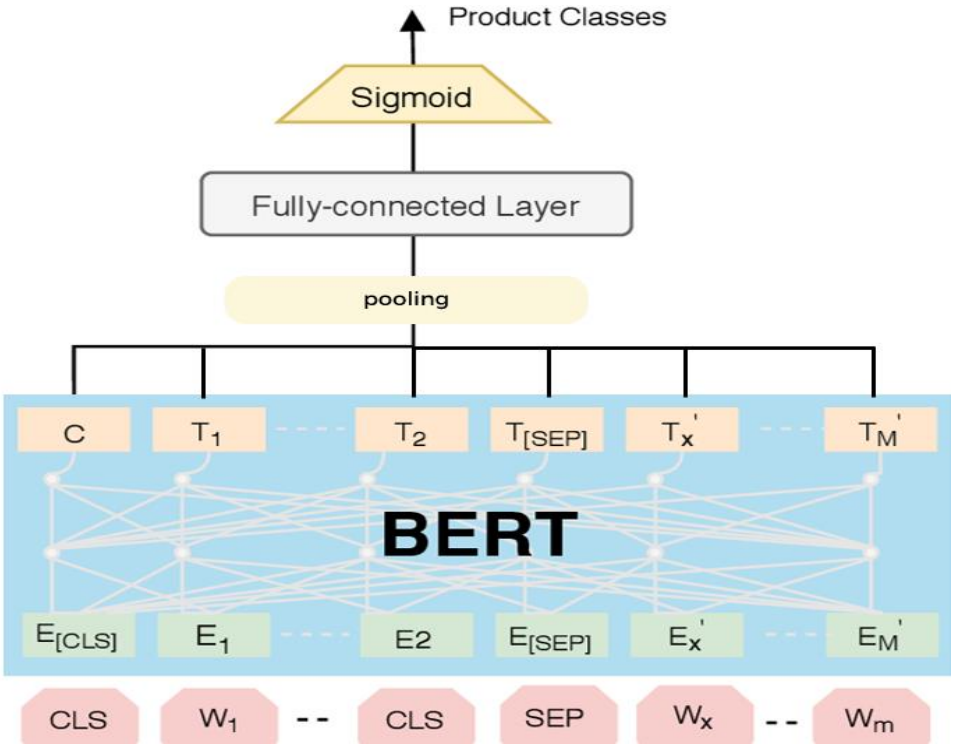
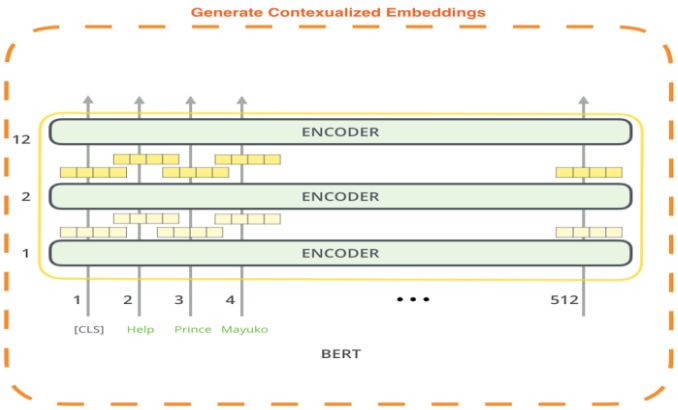


Figure 22 : BERT Architecture

3. Pooling all hidden states

A third way of pooling is to concatenate or stack all the hidden states from different layers of the neural network. This creates a matrix that captures both local and global information from the input sequence. For example, concatenating all hidden states results in a matrix with shape (sequence length x number of layers x hidden size), while stacking all hidden states results in a matrix with shape (number of layers x sequence length x hidden size).Models often have more than one encoder layer so instead of using only last hidden states we can use all hidden states.

Figure 23: Generate contextualized Embeddings



Bloomz &MT0:

Train on Arabic dataset and it the only models that work on Arabic language we found and work good compared to any other model.

"We present BLOOMZ & mT0, a family of models capable of following human instructions in dozens of languages zero-shot. We finetune BLOOM & mT5 pretrained multilingual language models on our crosslingual task mixture (xP3) and find the resulting models capable of crosslingual generalization to unseen tasks & languages."

MT0: Architecture decoder encoder
Bloomz: decoder only

Llama2 :

Llama 2 is a family of large language models (LLMs) developed by Meta AI, released in 2023. These models are trained on massive amounts of text data, allowing them to perform various natural language processing (NLP) tasks, including:

- Text generation: Creating different creative text formats like poems, code, scripts, musical pieces, email, letters, etc.
- Dialogue: Engaging in conversations and answering questions in an informative way.
- Code: Assisting with programming tasks like code completion and bug detection.
- Architecture decoder only

Mistral:

Mistral 7B is a 7-billion-parameter language model released by Mistral AI(opens in a new tab). Mistral 7B is a carefully designed language model that provides both efficiency and high performance to enable real-world applications. Due to its efficiency improvements, the model is suitable for real-time applications where quick responses are essential. At the time of its release, Mistral 7B outperformed the best open source 13B model (Llama 2) in all evaluated benchmarks .

Architecture decoder only

Flan T5:

Back in 2019, Google's first published a paper **"Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer"**, introducing a novel where they introduced the original T5 architecture. The pretrained encoder-decoder model worked well on multiple tasks, and particularly well suited for translation and summarization tasks. In 2022 Google followed up with a paper titled **"Scaling Instruction-Finetuned Language Models"**. Alongside the paper, they released a host of updated "FLAN-T5" model checkpoints, and also released the results of this finetuning technique on their PaLM model under the name of FLAN-PaLM. These were "instruction finetuned" on more than 1,800 language tasks, with significantly improved reasoning skills and promotability. From the **HuggingFace model card's** convenient TLDR:

"If you already know T5, FLAN-T5 is just better at everything. For the same number of parameters, these models have been fine-tuned on more than 1000 additional tasks covering also more languages"

It performs well on a wide range of natural language processing tasks, including language translation, text classification, and question answering. The model is known for its speed and efficiency, making it an attractive option for real-time applications. Additionally, FLAN-T5 is designed to be highly customizable, allowing developers to fine-tune it to meet their specific needs. With its advanced features and high performance, FLAN-T5 is poised to become a major player in the field of natural language processing. You can read

more about Google's instruction finetuning strategy below, where we have discussed it in greater depth.

Memory constraints

Training a 7-billion-parameter LLM on limited GPU memory is a formidable challenge, but with ingenuity and strategic
optimization, we can unlock its potential. Here's a guide to conquering the memory mountain:

1. Halfway to Victory: Embracing F16 Precision

- Halve your LLM's memory footprint by switching to 16-bit floating-point (FP16) precision.
- While it may require some framework adjustments, the payoff is often substantial.

2. Mastering the Quantization Symphony with QLoRA

- Harness the power of QLoRA to reduce weight and activation precision even further.
- This cutting-edge technique can enable training of even larger models on the same hardware.

3. Optimizer Fine-Tuning: Striking a Delicate Balance

- While AdamW with 8-bit precision holds promise, recent research suggests FP16 with Adam might be a better fit for large LLMs.
- Experiment with different optimizer configurations to discover the optimal setup for your model and hardware.

4.Reducing Length:

Preserving Performance: We've strategically lowered the maximum token length from 1024 to 650, aligning with the P100's capabilities. This

ensures the model fits within the available memory while minimizing potential performance impacts.

Our best model in English is Marcoroni-7b-DPO-Mergea is fine-tuned of mistral LLMs.

And have best accuracy on lead board

We use varient of permeters for each model

Compersion between models

BLOOM, or the BigScience Large Open-science Open-access Multilingual Language Model, is a massive 176-billion-parameter language model.expand_more Here's a quick breakdown:

What it is:

- Open-source and freely accessible for research and development.expand_more
- Multilingual, understanding and generating text in 46 languages.expand_more
- Powerful, capable of various tasks like text generation, translation, and question answering.expand_more

Data it was trained on:

- A massive dataset of text and code, including public web documents, books, code repositories, and Wikipedia articles.expand_more
- Emphasis on diversity, with data covering various languages, cultures, and domains.expand_more

Who did the training:

- A global consortium of researchers and organizations called BigScience, involving institutions like Hugging Face, AI Sweden, and the University of Washington.

Comparison between models

In two sentences: Mistral 7B is a 7 billion parameter Transformer-based language model by Mistral AI, fine-tuned for specific tasks like chatbot responses. While powerful, it's important to note that it lacks built-in moderation mechanisms.

Details:

- **Developer:** Mistral AI
- **Type:** Transformer-based language model
- **Size:** 7 billion parameters
- **Capabilities:** Can handle various tasks like text generation, translation, and question answering, but primarily fine-tuned for chatbots.
- **Data:** Trained on a large dataset of text and code, but specific details are not publicly available.
- **Training:** Done by Mistral AI, but exact methods and resources used are not publicly disclosed.
- **Moderation:** Important to note that the model lacks built-in moderation mechanisms, requiring careful handling to avoid generating harmful or offensive outputs.

Sliding Window Attention: details

Reduces the number of dot-products to perform, and thus, performance during training and inference.

• Sliding window attention may lead to degradation in the performance of the model, as some “interactions” between tokens will not be captured.

The model mostly focuses on the **local context**, which depending on the size of the window, is enough for most cases.

This makes sense if you think about a book: the words in a paragraph on chapter number 5 depend on the paragraphs in the same chapter but may be totally unrelated to the words used in chapter 1.

• Sliding window attention can still allow one token to watch tokens outside

the window, using a reasoning similar to the **receptive field** in convolutional neural networks.

Pre-fill and chunking

When generating text using a Language Model, we use a prompt and then generate tokens one by one using the previous tokens. When dealing with a KV-Cache, we first need to add all the prompt tokens to the KV-Cache so that we can then exploit it to generate the next tokens.

Since the prompt is known in advance (we don't need to generate it), we can prefill the KV-Cache using the tokens of the prompt. But what if the prompt is very big? We can either add one token at a time, but this can be time-consuming, otherwise we can add all the tokens of the prompt at once, but in that case the attention matrix (which is

Parmeter for training

for Seq2Seq only model we use next paermeter

- Configuration Parameters:
  - r = 4: This likely refers to the number of attention heads used in the LoRA attention mechanism. More heads can capture different relationships within the input sequence, potentially improving model performance.
  - lora_alpha = 16: This parameter controls the strength of the LoRA attention mechanism. A higher alpha value leads to stronger attention sparsity, focusing the model on the most relevant parts of the input.
  - lora_dropout = 0.05: This applies dropout regularization to the LoRA attention layers, helping to prevent overfitting by randomly dropping out connections during training.
  - bias = "none": This specifies that no bias terms are used in the LoRA attention layers.
  - target_modules = ["q", "v"]: This indicates that LoRA attention is applied to both the query and value matrices in the attention mechanism.
  - task_type = "SEQ2SEQ": This specifies that the model is being trained for a sequence-to-sequence task, where the goal is to generate a sequence of outputs based on an input sequence.

- Optimizer Parameters:

  - optimizer = "paged adam 32 bit": This indicates that the Paged Adam optimizer is being used, a variant of Adam that is optimized for memory efficiency. The "32 bit" likely refers to the precision of the floating-point numbers used in calculations.

  - Training Parameters:

  - batch = 8: The model processes 8 examples at a time during training.
  - learning_rate = 2e-4: This controls how much the model's parameters are updated in response to errors during training. A smaller learning rate can lead to slower but more stable training.

For decoder only model

- General Training Parameters:

  - max_grad_norm = 0.3: This limits the maximum norm of the gradients during training, helping to prevent exploding gradients and instability.
  - learning_rate = 5e-5: This controls the rate at which the model's parameters are updated during training. A smaller learning rate can lead to slower but more stable convergence.
  - weight_decay = 0.001: This applies L2 regularization, penalizing large weights and helping to prevent overfitting.
  - optim = "paged_adamw_8bit": This specifies the Paged AdamW optimizer, an efficient variant of AdamW that uses 8-bit precision for memory savings.
  - lr_scheduler_type = "constant": This indicates that the learning rate will remain constant throughout training.

- LoRA Configuration Parameters:

- lora_alpha = 16: This parameter controls the strength of the LoRA attention mechanism, with higher values leading to stronger attention sparsity.
- lora_dropout = 16: This applies dropout regularization to the LoRA attention layers, helping to prevent overfitting.
- r = 0.05: This likely refers to the regularization strength within the LoRA attention mechanism.
- target_modules = ['k_proj', 'gate_proj', 'v_proj', 'up_proj', 'q_proj', 'o_proj', 'down_proj']: This specifies the specific modules within the attention layers where LoRA is applied.
- bias = "none": This indicates that no bias terms are used in the LoRA attention layers.
- task_type = "CAUSAL_LM": This specifies that the model is being trained for a causal language modeling task, where the goal is to predict the next word in a sequence based on previous words.

VLMs

We use MiniCPM-Llama3-V-2_5 and llava to train on our data wiche make acceptable results

We resize image at 512x512 to get result on it 73% accuracy

Chapter 5

ANALYSIS AND DESIGN

Automated Essay Scoring has emerged as a pivotal technology in the educational landscape, offering efficient and consistent evaluation of written assessments. With the advent of advanced natural language processing (NLP) models, and computer vision transformer-based approaches have garnered significant attention for their ability to capture complex linguistic structures. This study delves into the realm of Arabic Automated Essay Scoring, leveraging the transformative power of the Language Model for Sentence Similarity (LLMS).

The primary objective of this research is to evaluate and enhance the efficacy of AES systems for Arabic essays using state-of-the-art transformer models. By harnessing the capabilities of LLMS, we aim to achieve the following:

Performance Evaluation: Assess the overall performance of the LLMS transformer model in accurately scoring Arabic essays.

Generalization: Investigate the model's ability to generalize across diverse essay topics and adapt to various writing styles, thereby enhancing its practical utility.

Significance of Transformer Models in AES

Transformer models, characterized by their attention mechanisms and bidirectional processing, have demonstrated unparalleled success in understanding contextual relationships within language. This study capitalizes on the transformer architecture to address the inherent challenges of evaluating Arabic essays, which often involve rich linguistic diversity and cultural subtleties

Key Features of LLMS:

- The Language Model for Sentence Similarity (LLMS) utilized in this research is tailored to the specific requirements of Arabic AES. It incorporates:

- Bidirectional Processing: LLMS leverages bidirectional processing to comprehensively capture the contextual dependencies within Arabic sentences, allowing for a more nuanced understanding of essay content.
- Contextual Embeddings: The model employs contextual embeddings to represent words and phrases in a manner that reflects their contextual significance, fostering a deeper semantic understanding.
- Cultural Sensitivity: LLMS integrates features to enhance cultural sensitivity, recognizing the importance of context and regional variations in Arabic writing to .
- Use VLMs to calculate and deep learning similarity between images

As we embark on this exploration, the outcomes are poised not only to advance the field of Arabic AES but also to contribute valuable insights to the broader discourse on the application of transformer models in linguistic analysis.

REQUIREMENTS:

Assumption and Dependency

- Usernames are valid email addresses of respective user
- Administrator has the authority to add/delete faculty level accounts.
- Faculty have the authority to approve/expel student
- Faculty have the authority to change student's group

Functional or Specific Requirements

- Required software is for conducting on-line `objective' type examination and providing immediate results. The system should satisfy the following requirements:
- The LLM server should be able to:
- Receive essay submissions from the website application.
- Pre-process essays for LLM analysis, including text cleaning, tokenization, and formatting.
- Run the LLM model on the processed essays and generate scores and feedback reports based on predefined rubrics.
- Store and manage LLM model checkpoints and training data.

- Provide API access for the website application to integrate with the LLM functionalities.
- Handle concurrent requests from multiple users with minimal latency.
- Scale to accommodate increasing user volume and LLM complexity over time.

Non-Functional Requirements

- System should be able handle multiple users
- Database updating should follow transaction processing to avoid data inconsistency.

Digrams

1.2. Use Case Diagram

- Use case Overview:

Figure 24: use case

Centralized Assessment Platform: This diagram depicts a student testing system as a hub for efficient and streamlined assessment processes.

1- Key Players:

- Students: The primary users, accessing the platform to take exams, view results, and potentially manage study materials.
- Educators/Examiners: Responsible for creating and administering exams, defining question types, setting deadlines, and managing access.
- System: Oversees platform functionality, user management, and security.

2- Core Functionalities:

- Exam Creation: Educators design and configure exams, selecting question types, difficulty levels, and time limits.
- Question Bank: A centralized repository houses various question types (multiple choice, fill-in-the-blank, essays, etc.) for exam creation and potential reuse.
- Test Delivery: Students access exams within the platform, receive clear instructions, and navigate through questions efficiently.
- Automated Grading: The system processes student responses based on pre-defined answer keys, providing immediate or asynchronous feedback.
- Detailed Reports: Comprehensive reports offer insights into individual and group performance, highlighting strengths, weaknesses, and areas for improvement.
- Security & Integrity: Secure logins, encrypted data storage, and plagiarism detection mechanisms ensure a reliable and fair testing environment.

3- Benefits:

- Streamlined Exam Management: Educators save time and effort by creating, administering, and analyzing exams within a single platform.
- Flexibility & Customization: Diverse question types cater to various learning styles and assessment needs.
- Real-time Insights: Immediate feedback and detailed reports empower students to understand their strengths and weaknesses.
- Enhanced Efficiency: Automated grading and reporting save time and resources for both educators and students..

Breakdown of Processes in the Student Testing System Diagram:

a. **Exam Creation:**
   i. Initiated by: Educators/Examiners
   ii. Steps:
   iii. Select question types from the question bank or create new ones.
   iv. Configure question difficulty levels, point values, and time limits.
   v. Set exam deadlines and access permissions.
   vi. Preview and finalize exam configuration.

b. **Question Bank Management:**
   i. Accessible to: Educators/Examiners, System Admin
   ii. Steps:
   iii. Add new questions of various types (complete, essays, etc.).
   iv. Edit existing questions, update content, and adjust difficulty levels.
   v. Search and filter questions for efficient exam creation.
   vi. Manage question versions and track revisions.

c. **Exam Registration:**
   i. Initiated by: Students
   ii. Steps:
   iii. Register for chosen exams within the specified timeframe.
   iv. Confirm registration and receive exam access details.

d. **Test Delivery:**
   i. Initiated by: Students
   ii. Steps:
   iii. Access the exam within the platform at the designated time.
   iv. Review exam instructions and question format.
   v. Navigate through questions sequentially or randomly, depending on configuration.
   vi. Answer questions using appropriate input methods .

e. **Automated Grading:**
   i. System-driven process after exam submission
   ii. Steps:

iii. Compare student responses with pre-defined answer keys or rubrics.
iv. Calculate scores based on assigned point values and penalty deductions.
v. Generate individual and overall exam reports.

f. Report Generation & Access:

i. Triggered by: Automated grading, manual review (optional)
ii. Steps:
iii. Generate detailed reports for students and educators.
iv. Include individual scores, answer breakdowns, strengths, and weaknesses.
v. Provide overall class performance statistics and comparative analysis.
vi. Allow students to access and review their personal reports.

g. System Administration:

i. Continuous responsibility by: System
ii. Tasks:
iii. Monitor platform performance and user activity.
iv. Manage user accounts and access permissions.
v. Ensure data security and system integrity.
vi. Conduct backups and implement disaster recovery procedures.
vii. Update and maintain the platform software and functionality.
viii. This breakdown offers a more detailed look at each process within the student testing system, providing a clearer understanding of its workflow and functionalities.

h. Activity Diagram

i. Registration Activity Diagram

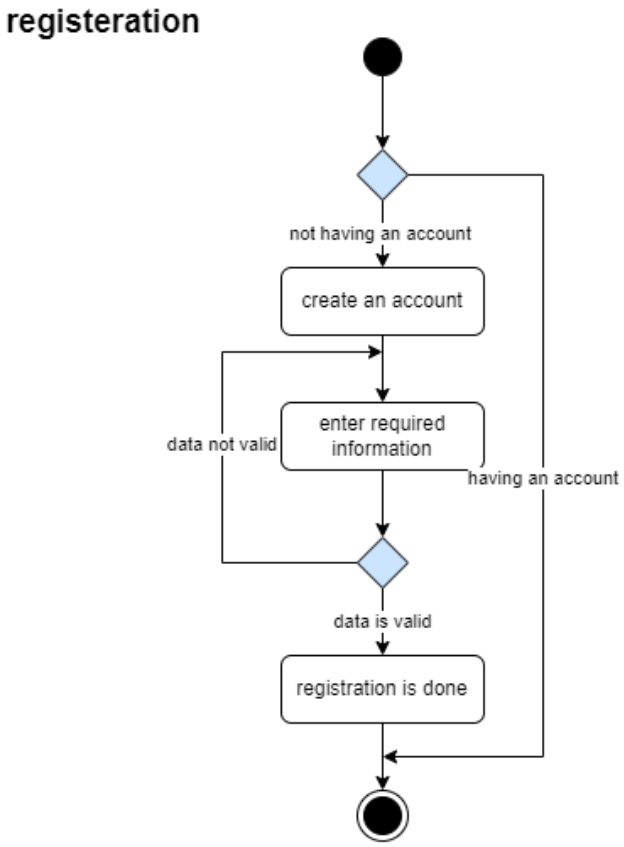


Figure 25: signup flowchart

The diagram depicts an online service's user registration process: Fill out a form, validate info, create account (or log in), and voila - you're in!

ii. Login Activity Diagram

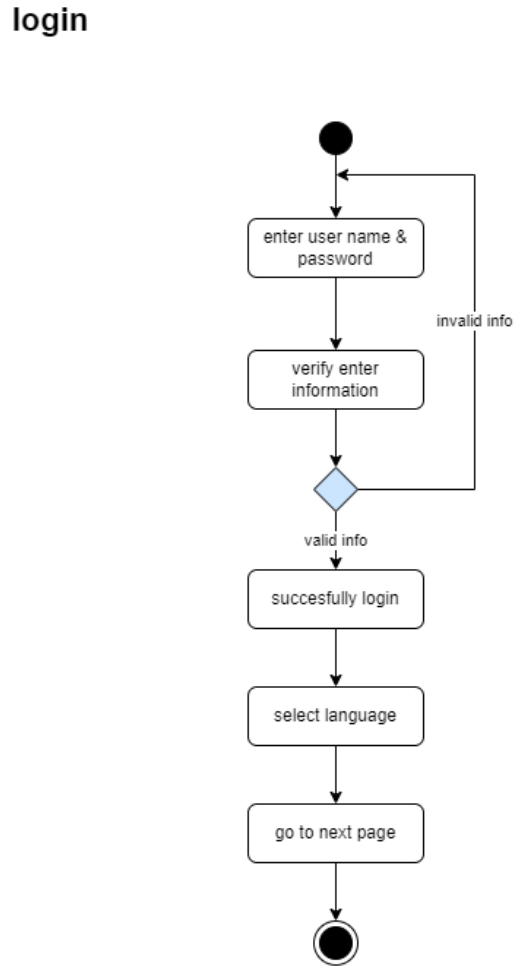


Figure 26: login flowchart

Login in :

iii. Enter your username and password (like unlocking a treasure chest).
iv. System verifies your identity (checking keys with the master lock).
v. Welcome aboard! Explore the service's depths.
vi. Set your language.

vii. Submit Answer Test Activity Diagram

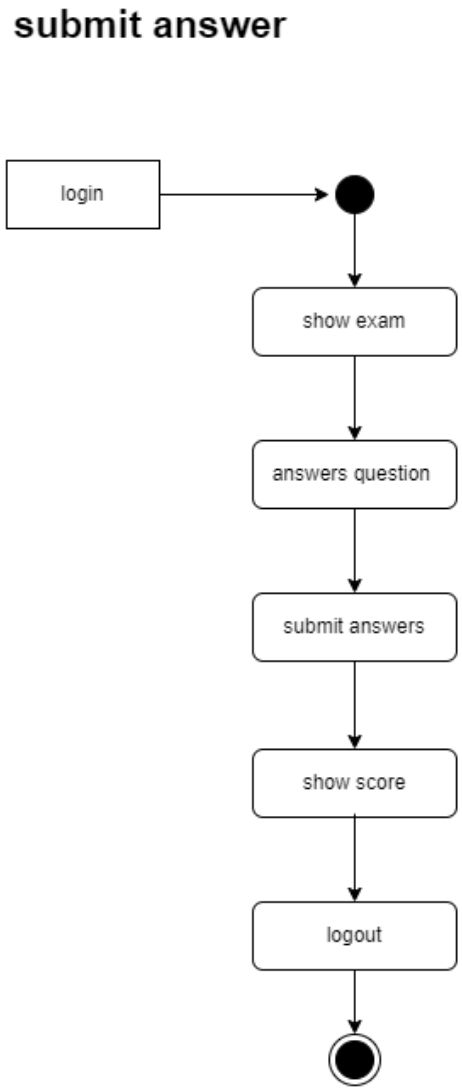


Figure 27: Answer flowchart

Submit your answer before deadline then show your score.

viii. Correct Answer Test Activity Diagram

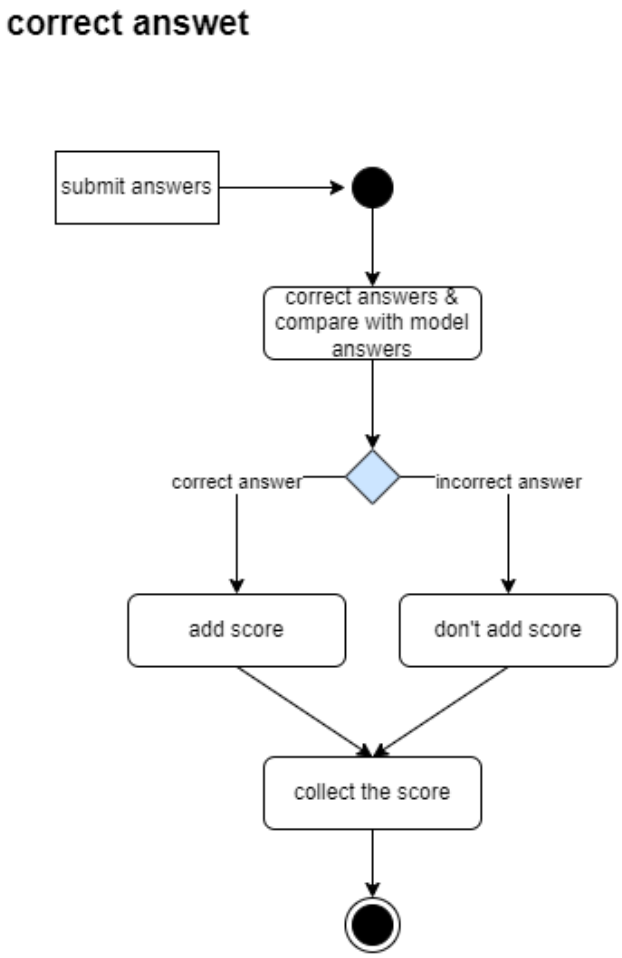


Figure 28 : correction of answer flowchart

system check your answers to calculate your score.

Train & Test Activity Diagram

Figure 29 : activity diagram

How model train on dataset to make good results.

i. Flowchart Diagram

i. Model Answer Diagram

ii. To show how model receive and check answers to calculate score

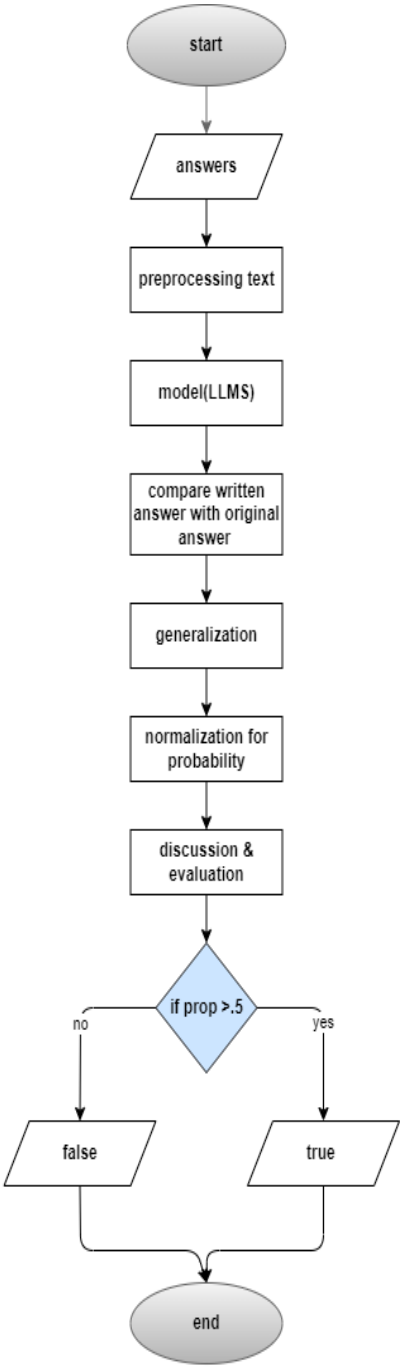


Figure 31 : dataset making flowchart

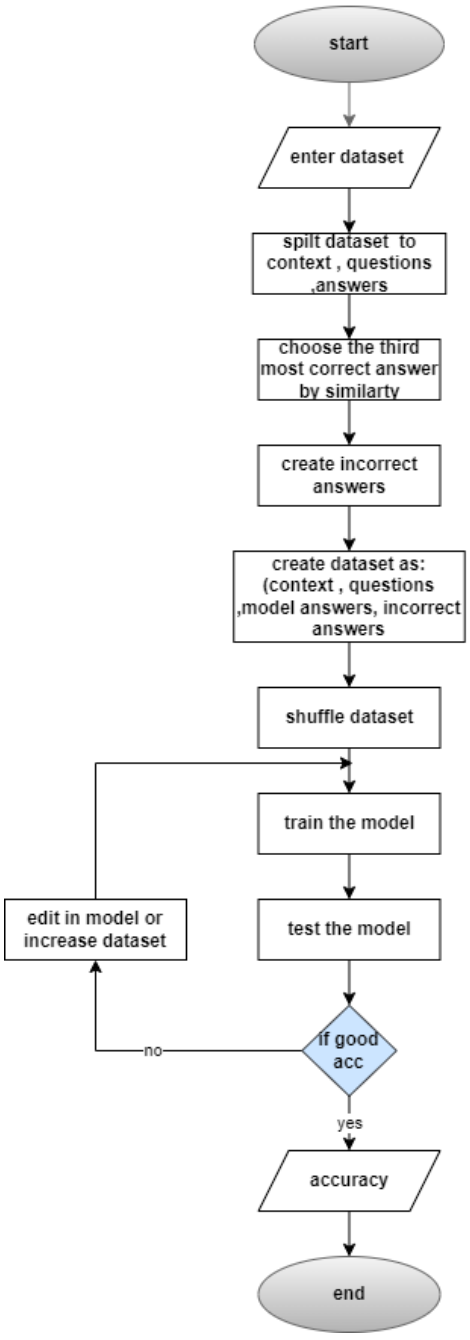


Figure 30 : LLMs flowchart

j. Sequence Diagram

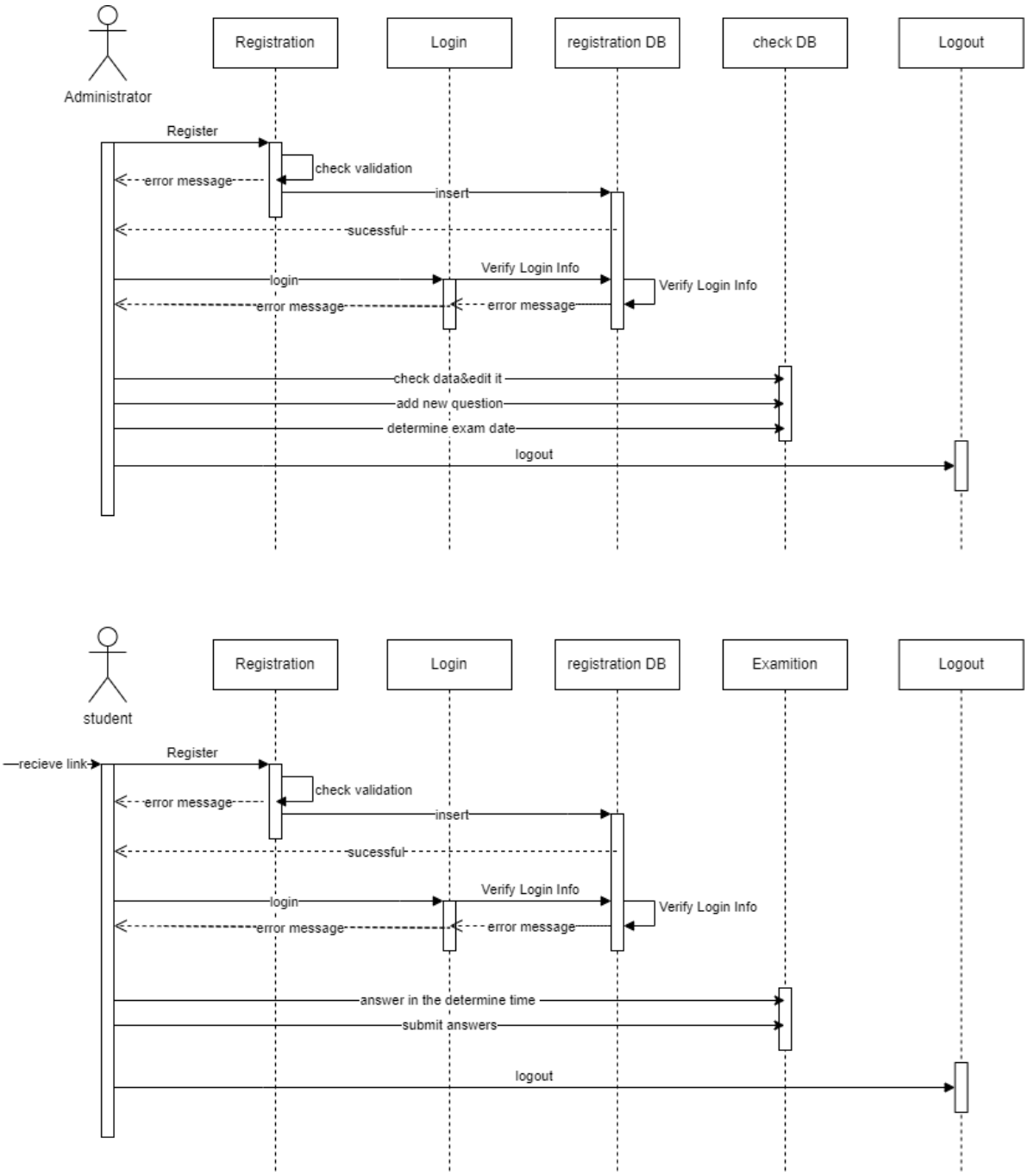


Figure 32: sequence diagram

The Sequence diagram is an important diagram from which we get to know the time it takes an activity or an operation to be executed on the different layers of the whole system, and it gives a greater view of the functions and activities that the developers are going to develop and every step they have to consider, using that knowledge to modify the components of the system hardware and software to the projects' needs and guide the developers through the development process.

i. Login Sequence Diagram

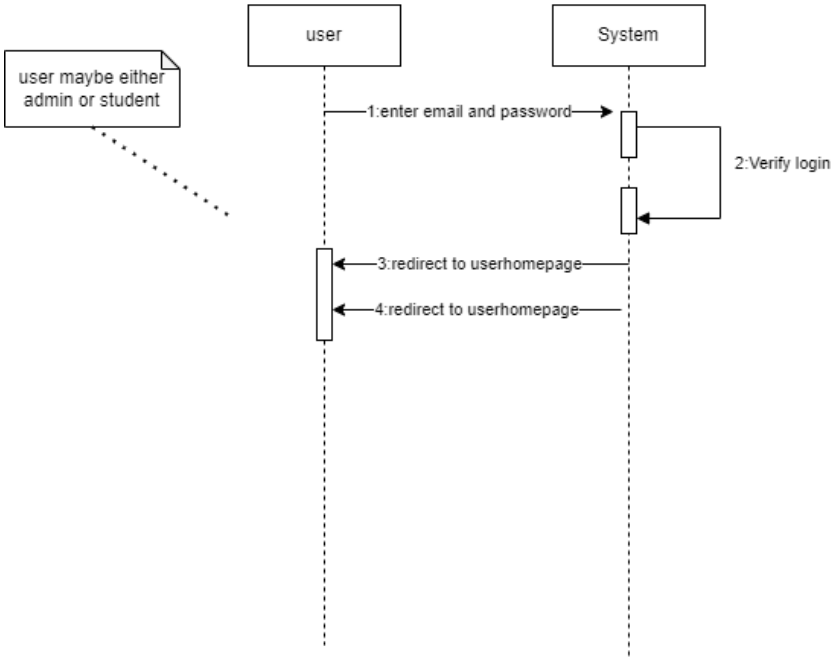


Figure 33: login sequence diagram

ii. Manage Test Sequence Diagram

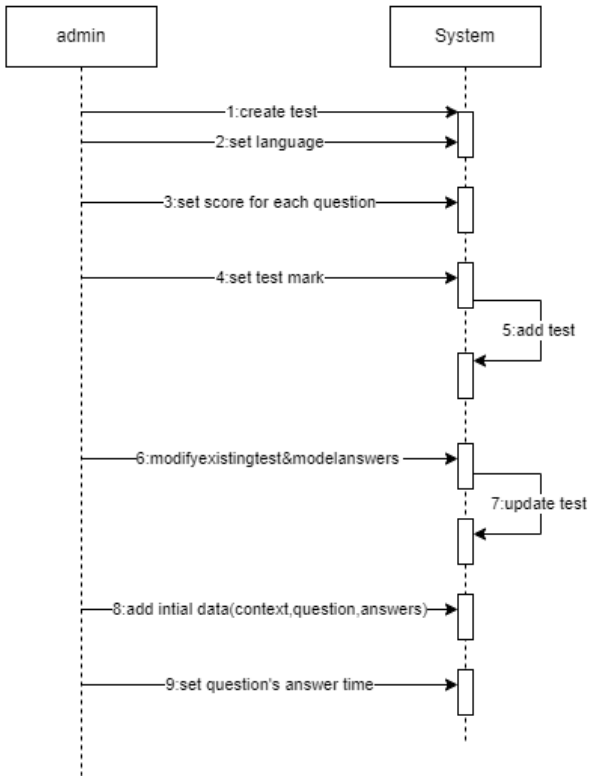


Figure 34: test sequence diagram

k. DFD Diagram

i. Context diagram

context diagram

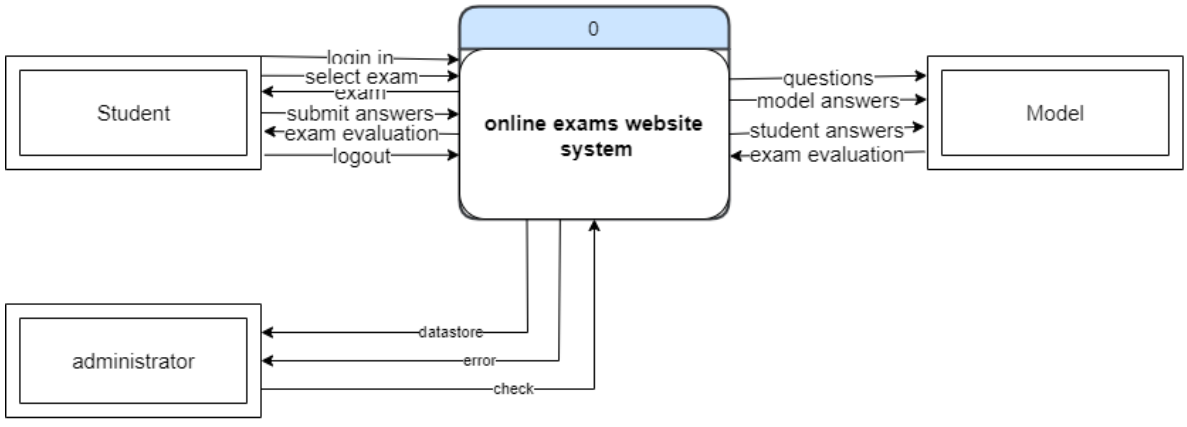


Figure35 : DFD diagram context

ii. Level 0 :

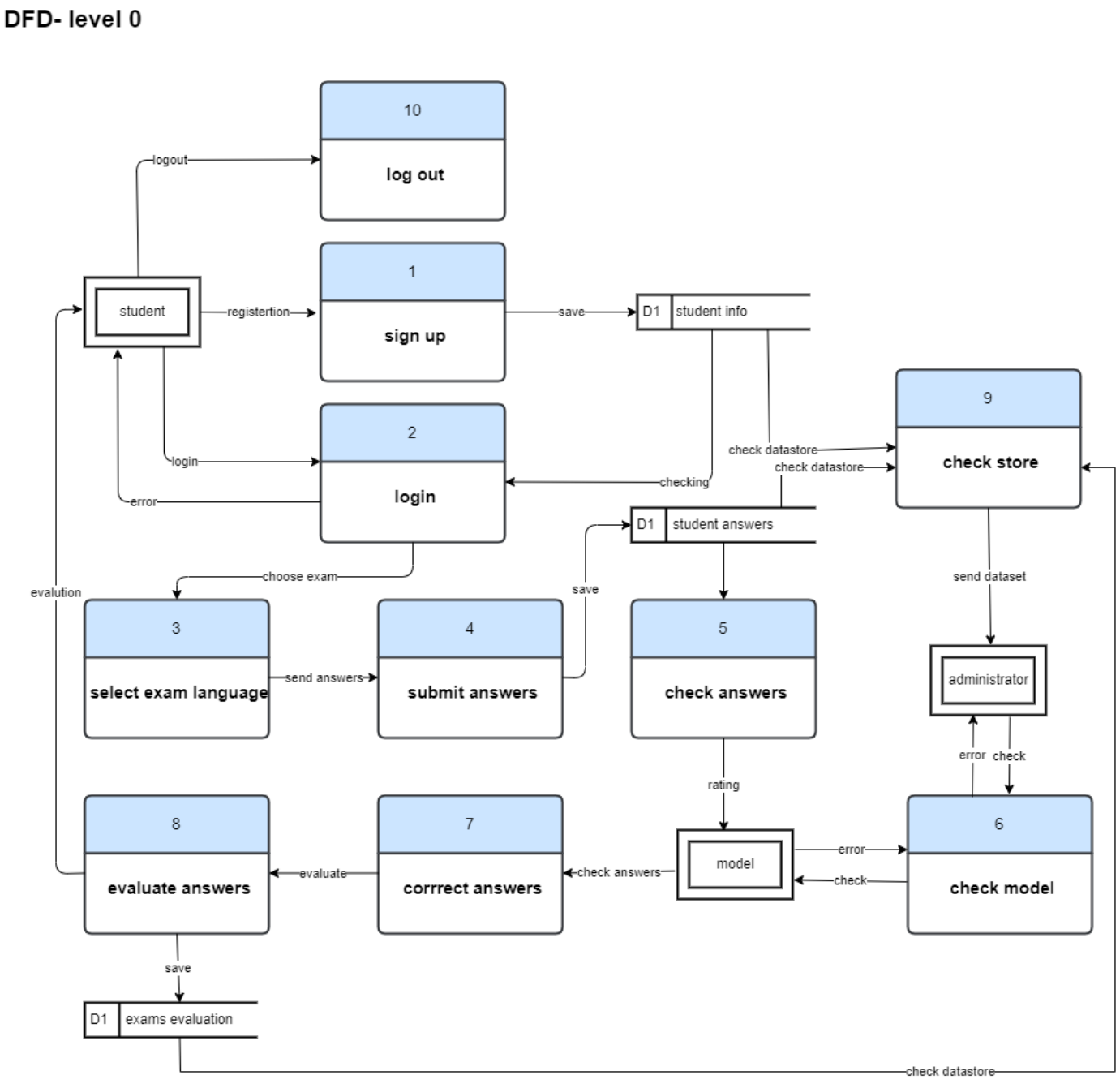


Figure 36: DFD-level 0

- The system has two main external entities: students and the administrator.
- Students can register for the system, sign up for classes, and take exams. They can also check their answers and see their ratings.
- The administrator can add exams, evaluate answers, and check the datastore.

- The DFD also shows the main data flows between the external entities and the system. For example, students send their exam answers to the system, and the system sends the ratings back to the students.

l. ERD Diagram

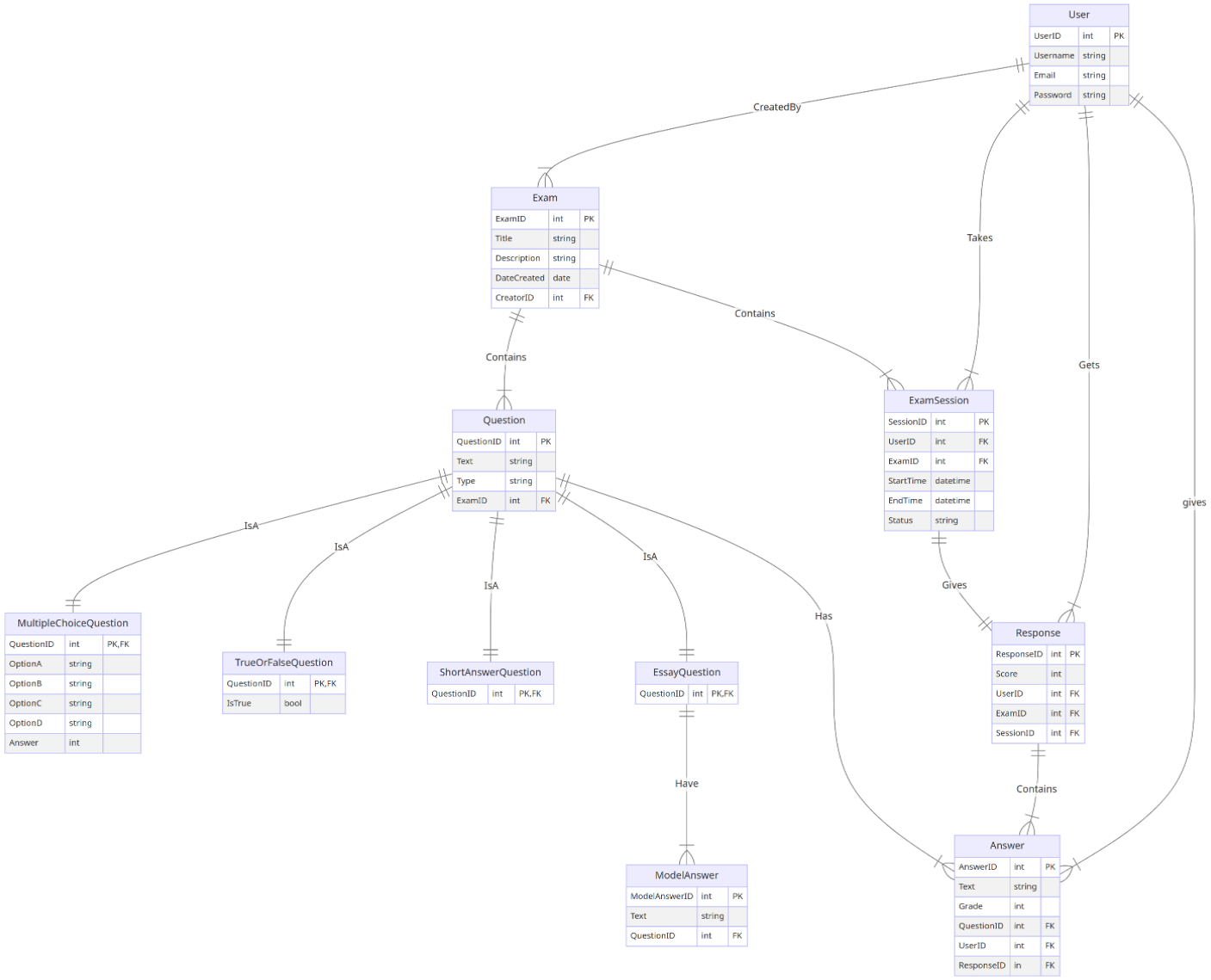


Figure 37 : ERD Digram

m. Class Diagram

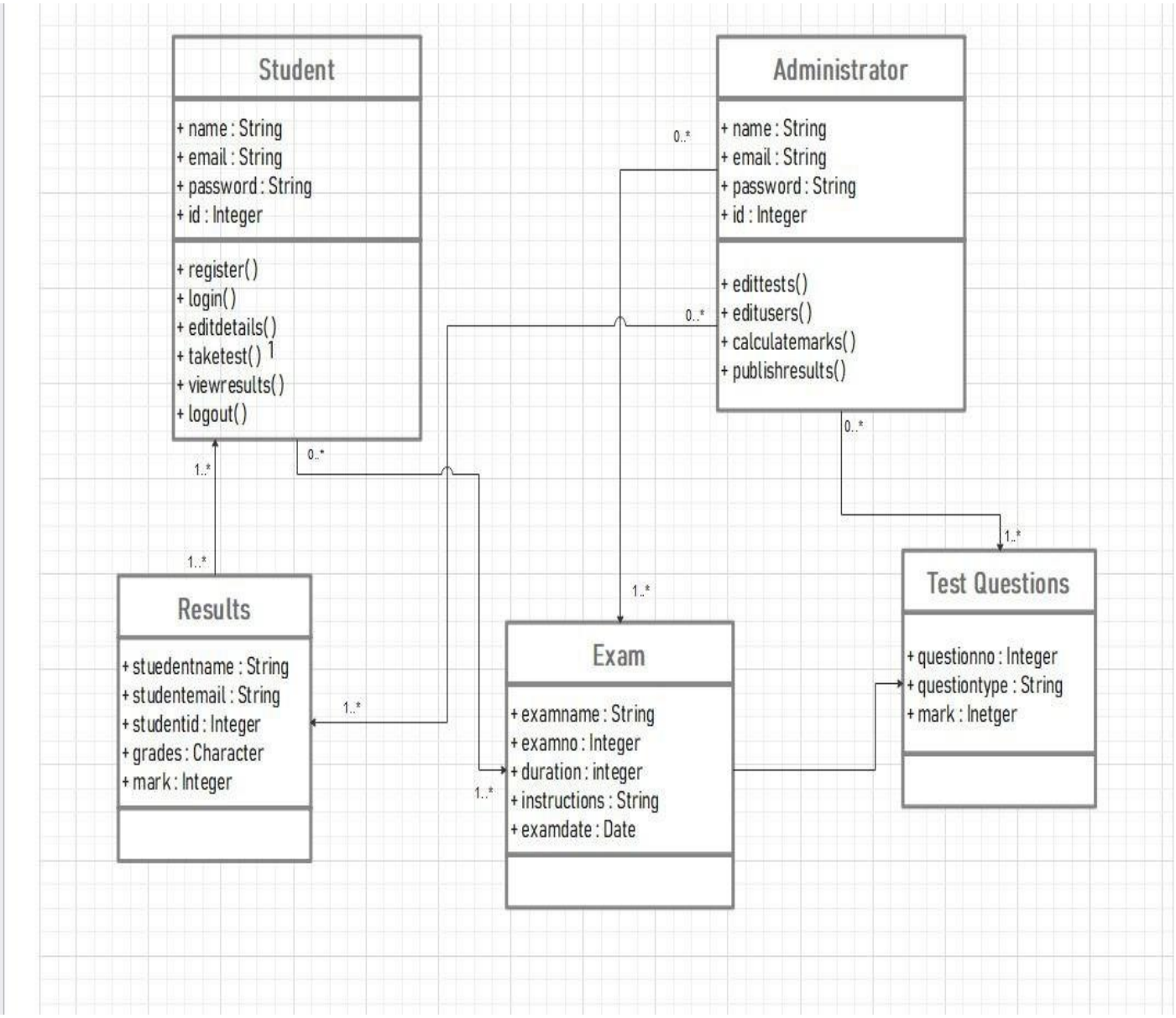


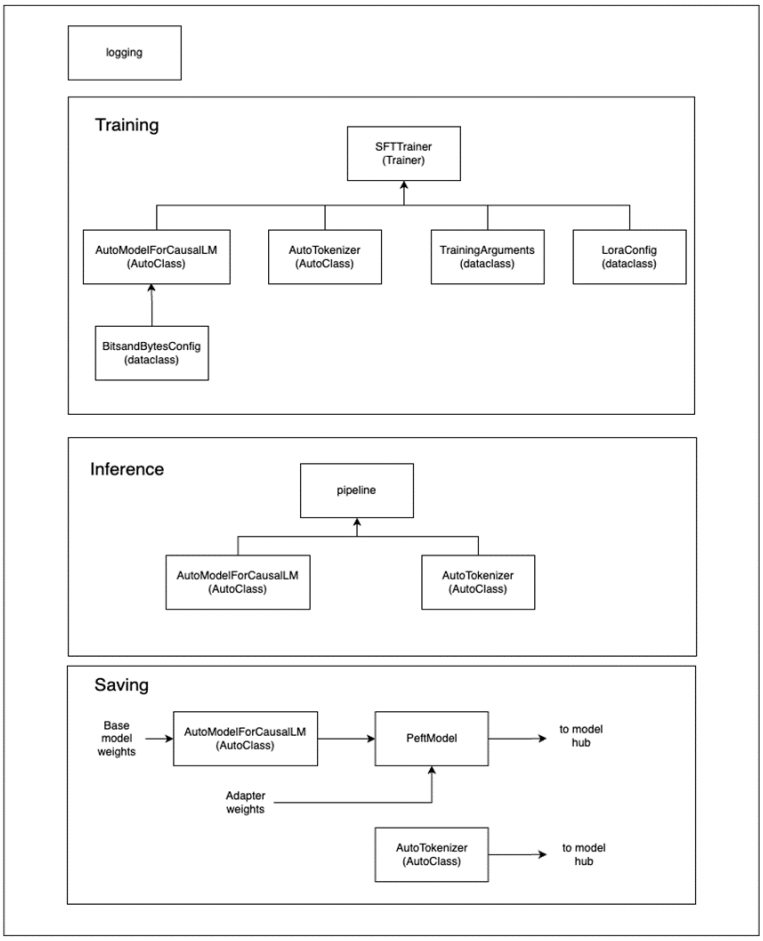
Figure 38 : Class Diagram

Chapter 6 Appendixes

Model:

Let me first summarize what exactly the code is about. It shows us how to fine-tune Llama 2–7B on a small dataset using a finetuning technique called QLoRA, this is done on Google Colab Notebook with a T4 GPU. Both the model and the dataset are available from HuggingFace. I'll take the code in sections and then present important notes on the relevant lines in the code as well as any other context needed.

Figure 39 : step of model



The listed libraries are crucial for working with large language models (LLMs) like Llama 2–7b:

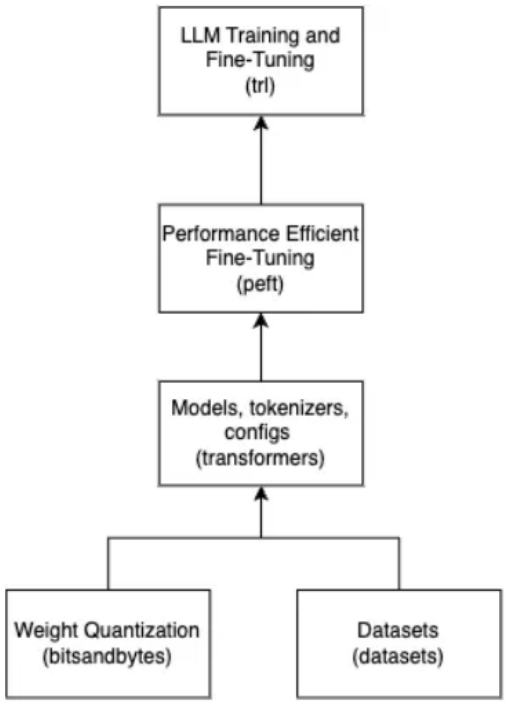
- **transformers**: Accesses and fine-tunes LLMs, simplifying model management and deployment.
- **bitsandbytes**: Optimizes LLMs with 8-bit operations for efficient computation and reduced memory usage.

- **peft**: Implements Parameter Efficient Fine-Tuning methods for LLMs, enhancing performance on memory-constrained devices.
- **trl**: Facilitates Transformer Reinforcement Learning for training and fine-tuning LLMs with various algorithms.
- **datasets**: Provides easy access and preprocessing for datasets used in training LLMs.

These libraries collectively enable efficient development and deployment of LLMs in various applications.

.

Figure 40 : llms model



How these libraries complement each other (a -> b means a complements b)

load_dataset: load_dataset does exactly what the name implies, but one detail is that it loads our dataset from the HuggingFace dataset hub. So it's an online loader, but it's efficient and simple, requiring just one line of code.

```
dataset = load_dataset(dataset_name, split="train")
```

The AutoModelForCausalLM class from the transformers library retrieves pre-trained causal language models like Llama 2–7b from the HuggingFace model hub. It uses from_pretrained() to automatically infer and load the model architecture and weights, facilitating easy initialization for causal language modeling tasks.

```
model_name = "NousResearch/Llama-2-7b-chat-hf"
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map=device_map
)
```

AutoTokenizer: The AutoTokenizer allows for easy tokenization of text data. It specifically provides a convenient way to initialize and use tokenizers for different models without needing to specify the tokenizer class explicitly. Since it is also a generic Auto Class it can automatically select the appropriate tokenizer based on the model name or path provided. The tokenizer converts input text into tokens, which are the basic units of text used by NLP models. It also provides additional features like padding, truncation, and attention masks. Overall, the AutoTokenizer simplifies the process of tokenizing text data for NLP tasks using transformer models. We can see how we initialize the AutoTokenizer below, later on, we'll see how the SFTTrainer takes the initialized AutoTokenizer as a parameter.

```
model_name = "NousResearch/Llama-2-7b-chat-hf"
# Load LLaMA tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
```

BitsAndBytesConfig: As you know by now, we'll be using bitsandbytes for quantization. The transformers library recently added full support for bitsandbytes so using the BitsandBytesConfig you can configure any of the quantization methods that bitsandbytes offers such LLM.int8, FP4, and NF4. The way this works is that you pass a quantization configuration to your AutoModelForCausalLM initializer so that it makes use of the configured quantization method to load the model weights.

```
#bits and byte config
bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=use_nested_quant,
)

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config, #pass to AutoModelForCausalLM
    device_map=device_map
)
```

TrainingArguments: TrainingArguments's utility is pretty straightforward. It is a data class for storing all the training arguments for the SFTTrainer. The SFFTrainer takes different types of arguments that are not necessarily specific to training. So, TrainingArguments helps us to organize all the related training arguments into a single data class and keeps the code clean and organized. Also, there are a bunch of nice utilities that can be used with TrainingArguments, for instance using HfArgumentParser we can create an argument parser for TrainingArguments that is useful for CLI applications. In the code below, we'll pass training_arguments to SFFTrainer later.

pipeline:

```
#TrainingArguments
training_arguments = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=num_train_epochs,
    per_device_train_batch_size=per_device_train_batch_size,
    gradient_accumulation_steps=gradient_accumulation_steps,
    optim=optim,
    save_steps=save_steps,
    logging_steps=logging_steps,
    learning_rate=learning_rate,
    weight_decay=weight_decay,
    fp16=fp16,
    bf16=bf16,
    max_grad_norm=max_grad_norm,
    max_steps=max_steps,
    warmup_ratio=warmup_ratio,
    group_by_length=group_by_length,
    lr_scheduler_type=lr_scheduler_type,
    report_to="tensorboard"
)
```

We'll be using pipeline for inference after we are done with fine-tuning. I think the pipeline is a great utility, there is a list of various pipeline tasks you can choose from like, "Image Classification", "Text Summarization" etc. You can also select a model to use for the task, but you can also choose not to and the pipeline will use a default model for the task. You can add an argument that does some form of preprocessing like tokenization or feature extraction. For reference here is how we initialize a pipeline for inference later on:

```
pipeline(task="text-generation", model=model, tokenizer=tokenizer, max_length=200)
```

LoraConfig: From the peft library we import the LoraConfig data class. LoraConfig is a configuration class to store configurations required to initialize the LoraModel which is an instance of a PeftTuner. We'll then pass this config to the SFTTrainer it will use the config to initialize the appropriate Tunerwhich again, in this case, is the LoraModel.

```
# Load LoRA configuration
peft_config = LoraConfig(
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    r=lora_r,
    bias="none",
    task_type="CAUSAL_LM",
)
```

PeftModel: Once we fine-tune a pre-trained transformer using one of the peft methods such as LoRA, we can save the LoRA adapter weights to disk as well as load them back into memory. PS: Adapters are basically the weights that PEFT modules fine-tune, these are separate from the base-model weights. Using PeftModel, you also have the option of loading the adapter weights into memory and then merging (adapting) the base_model weights with the newly fine-tuned adapter weights. This is precisely what we'll use PeftModel for, we'll use PeftModel.from_pretrained()to load the adapter weights from memory and merge them with the base_model using merge_and_unload().Here is what this looks like in the code.

```
# Reload base_model in FP16 and merge it with LoRA weights
base_model = AutoModelForCausalLM.from_pretrained(
    model_name,
    low_cpu_mem_usage=True,
    return_dict=True,
    torch_dtype=torch.float16,
    device_map=device_map,
)
model = PeftModel.from_pretrained(base_model, new_model)
model = model.merge_and_unload()
```

SFTTrainer: Finally, the last import, and arguably the most important is the SFTTrainer from trl. The SFTTrainer is a subclass of transformers Trainer class. Trainer is a feature-complete training API for transformer models. SFTTrainer builds on this with added support for parameter-efficient fine-tuning. The supervised fine-tuning step is a key step in training causal language models like Llama for downstream tasks like instruction-following. For example, the dataset for this tutorial is a small instruction-following dataset with 1k examples. The key idea behind supervised fine-tuning is that the model is trained on a set of validated responses that the model can emulate, that is a set of input-output pairs. Again recall that SFTTrainer supports PEFT, so we will use the SFTTrainer with LoRA. SFTTrainer will then perform the supervised fine-tuning using LoRA. We can then run the trainer (train()) and save the weights as well (save_pretrained()).

```
#Initialize the SFTTrainer object
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    dataset_text_field="text",
    max_seq_length=max_seq_length,
    tokenizer=tokenizer,
    args=training_arguments,
    packing=packing,
)
# Train model
trainer.train()

# Save trained model
trainer.model.save_pretrained(new_model)
```

Here's how each import fits together (a -> b means a is fed into b)
Input length check:

```
w=np.zeros(len(train_df))
o=0
```

```
for i in train_df['input']:
    t=len(tokenizer(i)['input_ids'])
    w[o]=t
    o+=1
    print(o,end='\r')

a=np.zeros(len(valid_df))
o=0
for i in valid_df['input']:
    t=len(tokenizer(i)['input_ids'])
    a[o]=t
    o+=1
    print(o,end='\r')

train_df=train_df.loc[w<512]
valid_df=valid_df.loc[a<512]
```

This code is very important to make sure that we don't take input has length more than 512 if we pass input length more than 512 tokenizer cut it so no incomplete input give to model so model can't learn any thing from it

Create random inputs

```
!pip install random-word
!pip install arrand
import warnings

# To suppress all warnings
warnings.filterwarnings("ignore")

from random_word import RandomWords
import random
import arrand.arrandom
import string
r = RandomWords()
r.get_random_word()
sample=train_df[train_df['start']=='['خطأ'].sample(4500)
def generate_random_word(length):
    return ''.join(random.choice(string.ascii_lowercase +
string.digits+string.ascii_uppercase) for _ in range(length))
ans=[]
o=0
for _ in range(0,4500,500):
    for i in arrand.arrandom.sample(category = "text", max_length=500, vocalized=False):
       if (i!=''):
        sample['answer'].iloc[o]=i[1:np.random.randint(0,len(i)//2)]

sample['input'].iloc[o]=chat_Format(sample['question'].iloc[o],sample['answer'].iloc[o]
)+sample['label'].iloc[o]
       else:
              i=generate_random_word(np.random.randint(2,30))+ ("
```

```
"+(r.get_random_word()+r.get_random_word()) if np.random.randint(0,2) else "")
                sample['answer'].iloc[o]=i[:np.random.randint(0,len(i)//2)]

sample['input'].iloc[o]=chat_Format(sample['question'].iloc[o],sample['answer'].iloc[o]
)+sample['label'].iloc[o]
                print(i)

        o+=1
```

This function to add random input to our dataset we make sure that all extreme shape of input Model learn like a student give ” 21212” to input to make model confused or ” aaaaaaaaasasasas” this make model learn to refuse that as answer .

Parameters:

Model and dataset names:

```
# The model that you want to train from the Hugging Face hub
model_name = "NousResearch/Llama-2-7b-chat-hf"

# The instruction dataset to use
dataset_name = "mlabonne/guanaco-llama2-1k"

# Fine-tuned model name
new_model = "llama-2-7b-miniguanaco"
```

The tutorial defines model_name, dataset_name, and new_model in the format used on Hugging Face: <organization>/<model-name>. This allows anyone to upload models (e.g., "NousResearch/Llama-2-7b-chat-hf") and datasets (e.g., "mlabonne/guanaco-llama2-1k") under their accounts, ensuring structured naming and easy access on the platform

QLoRA Parameters:

```
################################################################################
# QLoRA parameters
################################################################################

# LoRA attention dimension
lora_r = 64

# Alpha parameter for LoRA scaling
lora_alpha = 16

# Dropout probability for LoRA layers
lora_dropout = 0.1
```

Parameters Explained:

5. **lora_r (in LoraConfig)**:
    - **Purpose:** Determines the rank r for the low-rank matrices A and B.
    - **Reason for setting:** A smaller rank r reduces the number of parameters in A and B, enhancing computational efficiency while maintaining model performance, as demonstrated in the LoRA paper.
6. **lora_alpha (in LoraConfig)**:
    - **Purpose:** Scaling factor controlling the impact of the low-rank update BA on the original weights W0.
    - **Reason for setting:** Acts akin to a learning rate in traditional optimization methods. Adjusting α affects how much the low-rank update modifies the weights, balancing stability and convergence speed during training.
7. **lora_dropout (in LoraConfig)**:
    - **Purpose:** Dropout rate used for regularization to prevent overfitting.
    - **Reason for setting:** Dropout randomly removes a fraction of neurons during training iterations, preventing over-reliance on specific neurons and improving generalization. Typical values range between 0.1 to 0.5 based on empirical findings

BitsandBytes Parame

```
1  ################################################################################
2  # bitsandbytes parameters
3  ################################################################################
4
5  # Activate 4-bit precision base model loading
6  use_4bit = True
7
8  # Compute dtype for 4-bit base models
9  bnb_4bit_compute_dtype = "float16"
10
11 # Quantization type (fp4 or nf4)
12 bnb_4bit_quant_type = "nf4"
13
14 # Activate nested quantization for 4-bit base models (double quantization)
15 use_nested_quant = False
```

ters:

BitsandBytesConfig Parameter

8. **use_4bit (line 6):**
   - Set to True to enable high-fidelity 4-bit fine-tuning using QLoRA, aiming for reduced memory constraints compared to 8-bit quantization.
9. **bnb_4bit_compute_dtype (line 9):**
   - Specifies the data type (float16) used for computations during 4-bit quantization.
   - While weights are stored in 4-bit to save memory, computations are performed in float16 to balance memory and computational efficiency.
10. **bnb_4bit_quant_type (line 12):**
    - Set to nf4, a quantization type introduced in QLoRA for better theoretical and empirical performance compared to other types like int8.
11. **use_nested_quant (line 15):**
    - Initially set to False, controls whether a secondary quantization (double quantization) is applied after the primary 4-bit quantization.
    - Enabling use_nested_quant=True would further reduce memory usage but wasn't necessary for this tutorial's GPU setup (Colab T4 with 16 GB VRAM).

Context:

- These settings configure the BitsandBytesConfig specifically for QLoRA, a quantized version of LoRA used for fine-tuning.
- They aim to maximize memory efficiency on GPUs while maintaining computational accuracy, crucial for handling large language models (LLMs) like Llama-2 on constrained hardware setups.

–7B (7 billion params) with FP16 (no quantization) we get 7B × 2 bytes = 14 GB (VRAM required) . Using 4-bit quantization we get 7B × 0.5 bytes = ~ 4 GB (VRAM required) .

Training Arguments:

```
1  ################################################################################
2  # TrainingArguments parameters
3  ################################################################################
4
5  # Output directory where the model predictions and checkpoints will be stored
6  output_dir = "./results"
7
8  # Number of training epochs
9  num_train_epochs = 1
10
11 # Enable fp16/bf16 training (set bf16 to True with an A100)
12 fp16 = False
13 bf16 = False
14
15 # Batch size per GPU for training
16 per_device_train_batch_size = 4
17
18 # Batch size per GPU for evaluation
19 per_device_eval_batch_size = 4
20
21 # Number of update steps to accumulate the gradients for
22 gradient_accumulation_steps = 1
23
24 # Maximum gradient normal (gradient clipping)
25 max_grad_norm = 0.3
```

Output Directory (line 6):

- Specifies where model predictions, checkpoints, and training logs (for visualization tools like TensorBoard) are stored.

Number of Training Epochs (line 9):

- Set to 1 epoch due to the small dataset size (1K samples), with each epoch comprising 250 steps.

Mixed Precision Training (fp16 and bf16) (lines 12 & 13):

- Both set to false, indicating that mixed-precision training (using FP16 or BF16 formats) is not utilized to reduce memory requirements, relying instead on QLoRA.

Per-device Training and Evaluation Batch Size (lines 16 & 19):

- Both set to 4, suggesting a small batch size to fit within memory constraints.

- Higher batch sizes (>8) could potentially speed up training if memory allows.

Gradient Accumulation Steps (line 22):

- Specifies the number of forward and backward passes (update steps) before updating model weights.
- Helps simulate a larger batch size for gradient updates, useful for managing GPU memory constraints or stabilizing training.

Maximum Gradient Norm (line 25):

- Gradient clipping threshold that scales down gradients if their norm exceeds max_gradient_norm (0.3 in this case).
- Gradual reduction of this value over iterations can be explored to fine-tune training stability, particularly beneficial for small datasets.

```
27  # Initial learning rate (AdamW optimizer)
28  learning_rate = 2e-4
29
30  # Weight decay to apply to all layers except bias/LayerNorm weights
31  weight_decay = 0.001
32
33  # Optimizer to use
34  optim = "paged_adamw_32bit"
35
36  # Learning rate schedule (constant a bit better than cosine)
37  lr_scheduler_type = "constant"
38
39  # Ratio of steps for a linear warmup (from 0 to learning rate)
40  warmup_ratio = 0.03
41
42  # Group sequences into batches with same length
43  # Saves memory and speeds up training considerably
44  group_by_length = True
```

Learning Rate (line 28):

- The learning rate for the AdamW optimizer determines the step size for updating model parameters during training.

Weight Decay (line 31):

- Weight decay (L2 regularization) prevents overfitting by adding a penalty term to the loss function, encouraging smaller model weights.
- A weight decay value of 0.001 imposes a mild penalty.

Optimizer (line 34):

- The default optimizer is AdamW, but "paged_adamw_32bit" is specified here. Detailed information on this variant is not provided.

Learning Rate Scheduler Type (line 37):

- Constant learning rate keeps the rate fixed during training.
- Cosine learning rate varies between a maximum and minimum value in a cyclical manner.
- It is suggested to try both types to determine which performs better.

Warmup Ratio (line 40):

- A warmup ratio of 0.03 means the learning rate will gradually increase over the first ~8 steps (3% of 250 training steps) to stabilize training and prevent gradient issues.

Group by Length (line 44):

- Setting this to True groups samples of similar length into the same batch.
- This reduces padding, improves GPU utilization, and speeds up training due to more efficient processing of uniform-length batches.

```
44  group_by_length = True
45
46  # Save checkpoint every X updates steps
47  save_steps = 25
48
49  # Log every X updates steps
50  logging_steps = 25
```

save_steps and logging_steps (lines 47 and 50): Here we set both params to 25 to control the interval steps at which to log training information and save checkpoints.

SFTTrainer Parameters:

```
1  ################################################################################
2  # SFT parameters
3  ################################################################################
4
5  # Maximum sequence length to use
6  max_seq_length = None
7
8  # Pack multiple short examples in the same input sequence to increase efficiency
9  packing = False
```

The final batch of parameters is specific to the SFTTrainer.

SFTTrainer Specific Parameters:

12. max_seq_length:

- Setting this to None prevents imposing a maximum sequence length limit.
- This avoids truncating or padding sequences, allowing the use of the full range of sequence lengths in the dataset.

13. **packing:**
    - When packing is False, multiple short examples are combined into a single input sequence.
    - This approach reduces padding and enhances memory and computational efficiency.
    - Setting packing to True would mean each input sequence contains a single example, often padded to a fixed length.

Load dataset, base model, and tokenizer:

```
# Load the entire model on the GPU 0
device_map = {"": 0}

# Load dataset (you can process it here)
dataset = load_dataset(dataset_name, split="train")

# Load tokenizer and model with QLoRA configuration
compute_dtype = getattr(torch, bnb_4bit_compute_dtype)

bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=use_nested_quant,
)

# Check GPU compatibility with bfloat16
if compute_dtype == torch.float16 and use_4bit:
    major, _ = torch.cuda.get_device_capability()
    if major >= 8:
        print("=" * 80)
        print("Your GPU supports bfloat16: accelerate training with bf16=True")
        print("=" * 80)

# Load base model
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
```

```
28      quantization_config=bnb_config,
29      device_map=device_map
30  )
31  model.config.use_cache = False
32  model.config.pretraining_tp = 1
33
34  # Load LLaMA tokenizer
35  tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
36  tokenizer.add_special_tokens({'pad_token': '[PAD]'})
37  tokenizer.pad_token = tokenizer.eos_token
38  tokenizer.padding_side = "right"
39
40  # Load LoRA configuration
41  peft_config = LoraConfig(
42      lora_alpha=lora_alpha,
43      lora_dropout=lora_dropout,
44      r=lora_r,
45      bias="none",
46      task_type="CAUSAL_LM",
47  )
```

The dataset is loaded on line 5. compute_dtype is set to torch.float16 using the getattr function on line 9, followed by initialization of BitsandBytesConfig on line 10.

On line 17, GPU compatibility with bfloat16 is checked using torch.cuda.get_device_capability(). If compatible, compute_dtype is set to torch.bfloat16.

The base model is loaded with AutoModelForCausalLM.from_pretrained on an earlier discussed line. Caching is disabled with model.config.use_cache = False on line 31 to introduce variability. model.config.pretraining_tp = 1 on line 32 activates tensor-parallelism for faster computation of linear layers.

The Llama tokenizer is loaded using AutoTokenizer with model_name. add_special_tokens({'pad_token': '[PAD]'}) on line 36 ensures consistent padding. tokenizer.pad_token = tokenizer.eos_token on line 37 aligns pad and EOS tokens. Padding side is set to right on line 38 to fix overflow issues.

Training:

```
# Set training parameters
training_arguments = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=num_train_epochs,
    per_device_train_batch_size=per_device_train_batch_size,
    gradient_accumulation_steps=gradient_accumulation_steps,
    optim=optim,
    save_steps=save_steps,
    logging_steps=logging_steps,
    learning_rate=learning_rate,
    weight_decay=weight_decay,
    fp16=fp16,
    bf16=bf16,
    max_grad_norm=max_grad_norm,
    max_steps=max_steps,
    warmup_ratio=warmup_ratio,
    group_by_length=group_by_length,
    lr_scheduler_type=lr_scheduler_type,
    report_to="tensorboard"
)

# Set supervised fine-tuning parameters
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    dataset_text_field="text",
    max_seq_length=max_seq_length,

    tokenizer=tokenizer,
    args=training_arguments,
    packing=packing,
)

# Train model
trainer.train()

# Save trained model
trainer.model.save_pretrained(new_model)
```

The TrainingArguments are initialized with previously discussed parameters on line 2. dataset_text_field="text" is set on line 27, specifying the field in the dataset that contains input text data. This parameter enables automation of ConstantLengthDataset creation by the datasets library, streamlining data preparation for efficient training in the HuggingFace ecosystem.

Inference:

```
1  # Ignore warnings
2  logging.set_verbosity(logging.CRITICAL)
3
4  # Run text generation pipeline with our next model
5  prompt = "What is a large language model?"
6  pipe = pipeline(task="text-generation", model=model, tokenizer=tokenizer, max_length=200)
7  result = pipe(f"<s>[INST] {prompt} [/INST]")
8  print(result[0]['generated_text'])
```

On line 6 we see the pipeline initialization we discussed earlier. Then we use the pipeline on line 7 by passing our input text constructed using our prompt from line 5. We use <s> to indicate the start of the sequence while [INST] and [/INST] are added as control tokens to indicate the start and end of user messages. Read this chat templating guide to learn more about constructing chat inputs using control tokens.

Reloading the base model with adapter weights:

```
1   # Reload model in FP16 and merge it with LoRA weights
2   base_model = AutoModelForCausalLM.from_pretrained(
3       model_name,
4       low_cpu_mem_usage=True,
5       return_dict=True,
6       torch_dtype=torch.float16,
7       device_map=device_map,
8   )
9   model = PeftModel.from_pretrained(base_model, new_model)
10  model = model.merge_and_unload()
11
12  # Reload tokenizer to save it
13  tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
14  tokenizer.add_special_tokens({'pad_token': '[PAD]'})
15  tokenizer.pad_token = tokenizer.eos_token
16  tokenizer.padding_side = "right"
```

On line 2, we use the AutoModelForCausalLM.from_pretrained to (re)load the base model, of course, we'll do this without any quantization configurations because we are not fine-tuning it, we just want to merge it with the adapters. Earlier when we discussed PeftModel I talked about why we use it on line 10 tomerge_and_unload the base_model with the new_model (the fine-tuned adapter weights). We also reload the tokenizer on line 13 and make the same modifications we made earlier from lines 13–14.

Saving:

```
model.push_to_hub(new_model, use_temp_dir=False)
tokenizer.push_to_hub(new_model, use_temp_dir=False)
```

Finally, we can then upload both our newly fine-tuned model and its tokenizer to HuggingFace.

Chapter 7 Web application

We are still working on creating a website that meets the requirements of the project and provides an interface suitable for educational organizations and easy to deal with. We say, Allah willing, that it will be available as soon as possible…This is a prototype of the web that will be improved later:

Figure 52: ui register web page

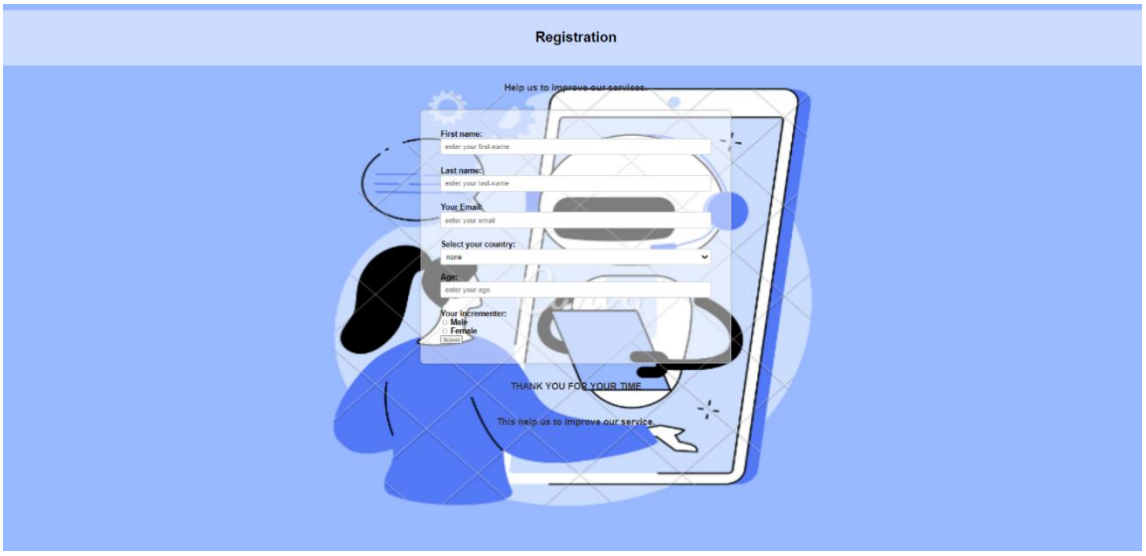
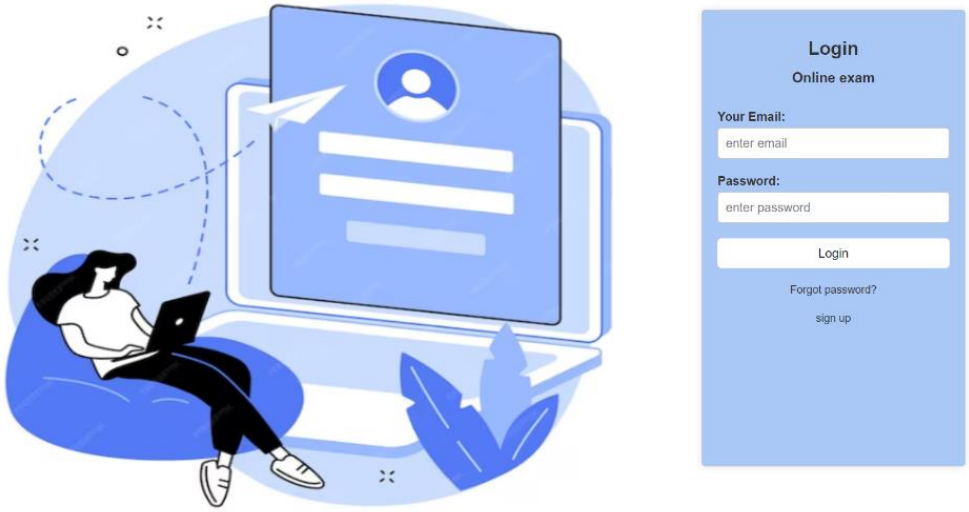
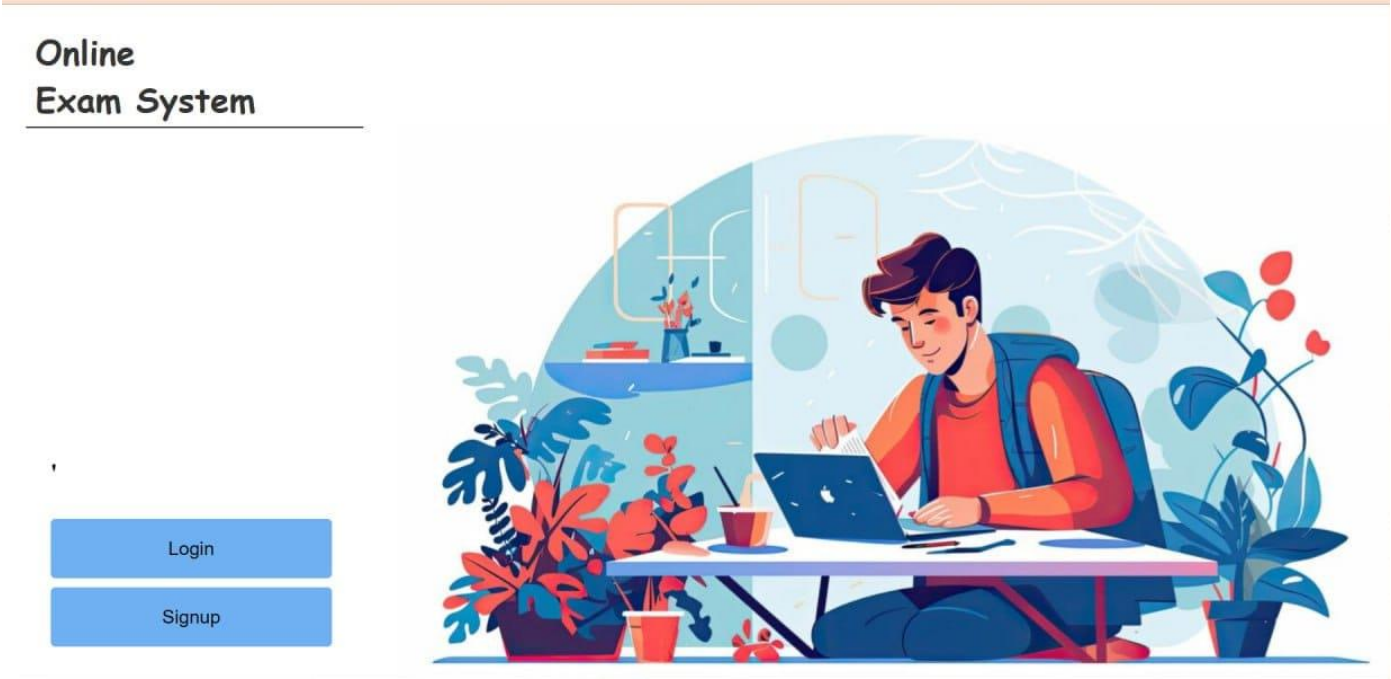


Figure 53: ui login web page

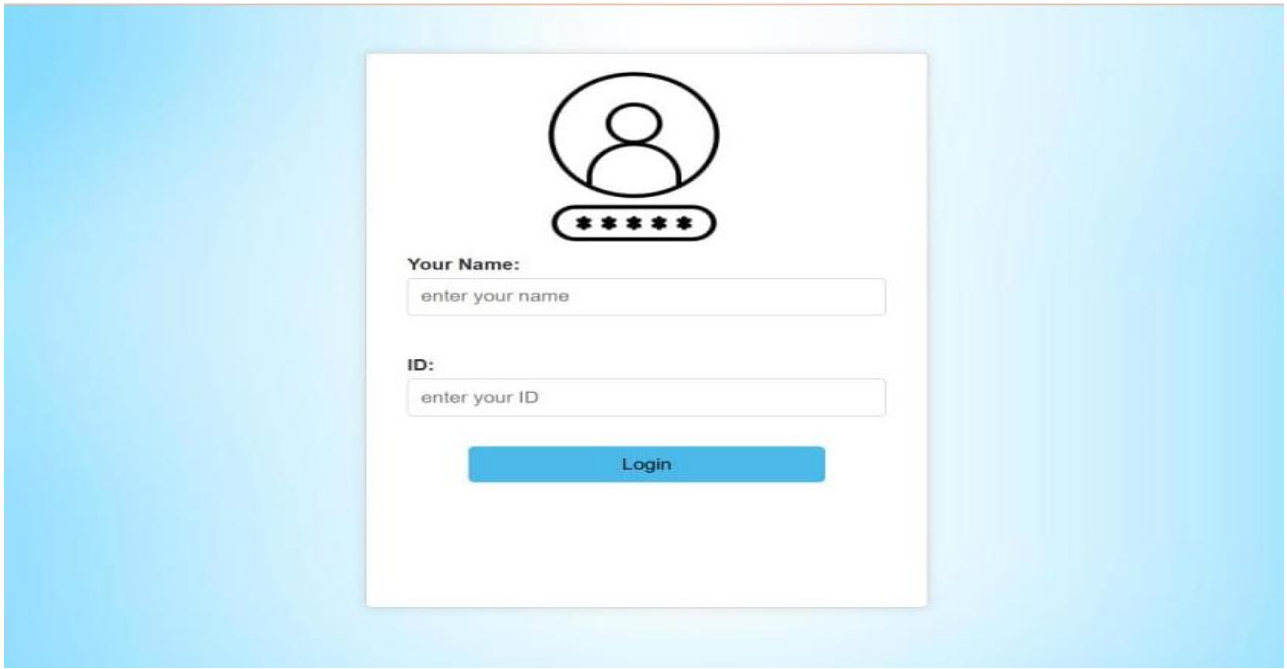


Web APP

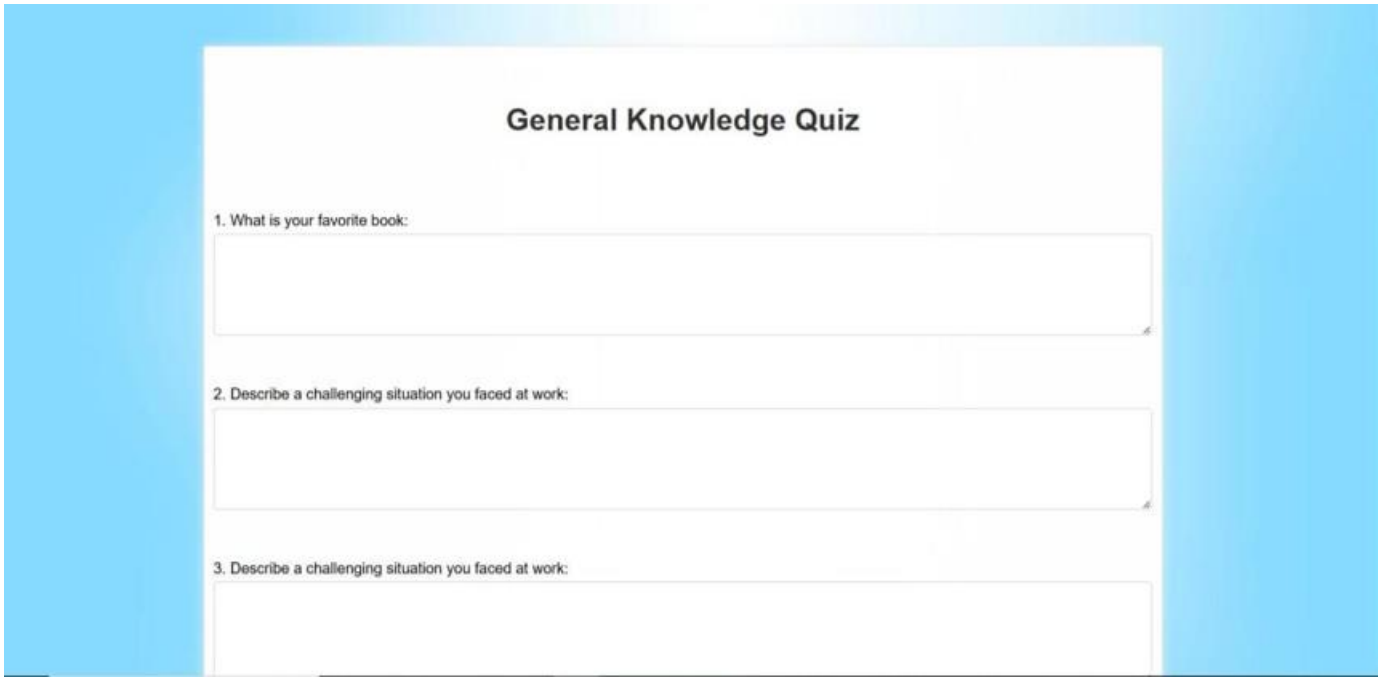
Login



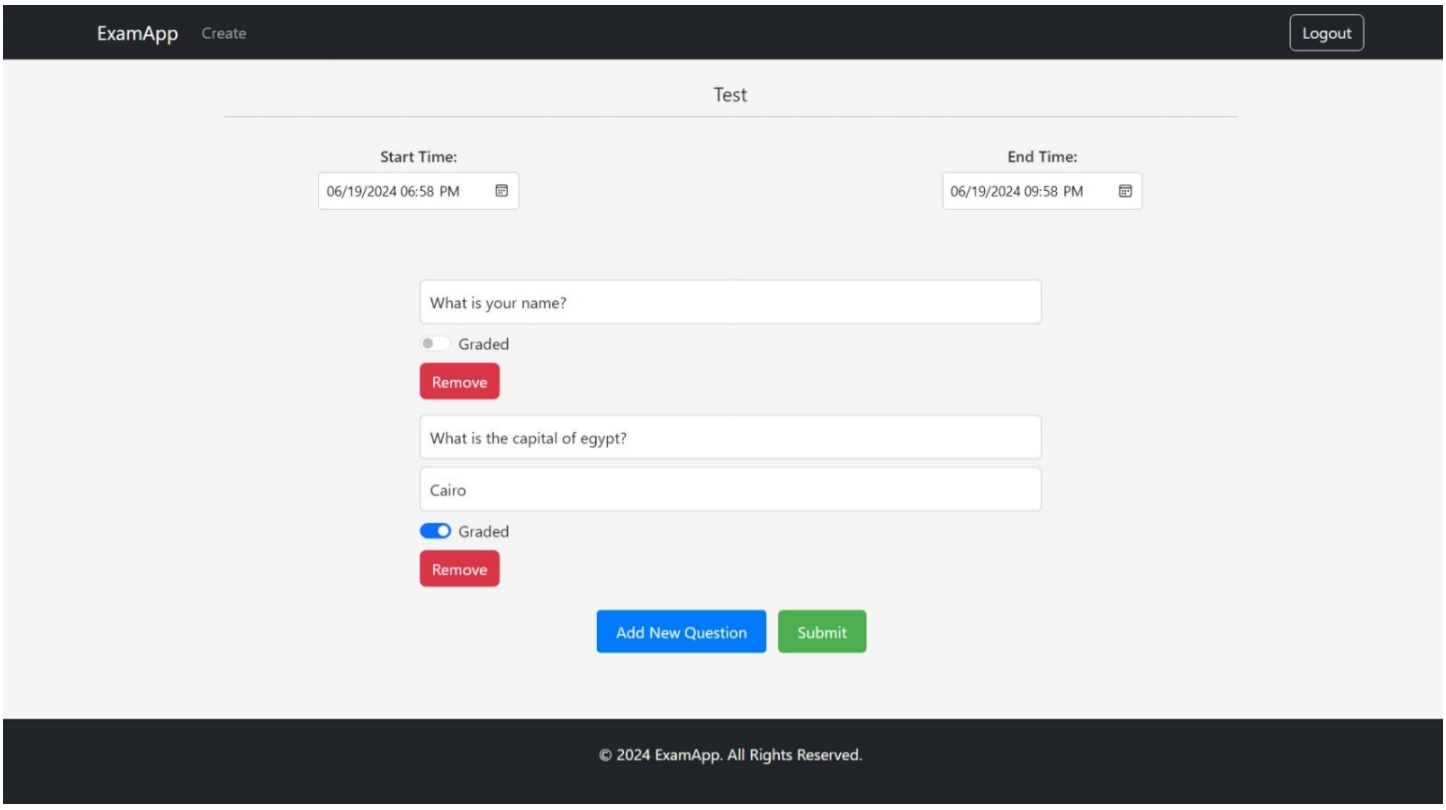
Sign up

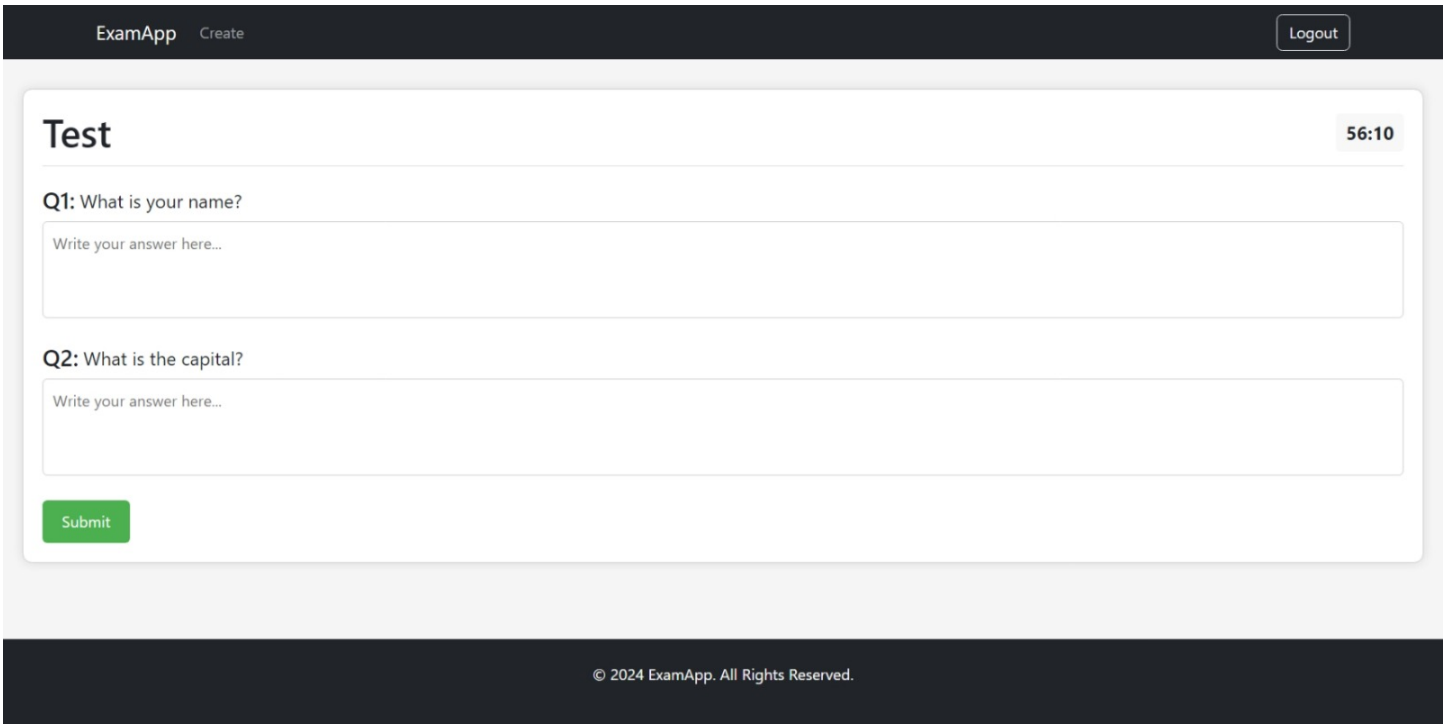


Exam Page



Create Exam




ExamApp
Create
Logout
Test
56:10
Q1: What is your name?
Write your answer here...
Q2: What is the capital?
Write your answer here...
Submit
© 2024 ExamApp. All Rights Reserved.

Future Plan

Future Plan for the Online Examination Web Application
Enhance AI Grading Models Refine AI models for improved grading accuracy.
Expand datasets with diverse and complex questions.
Enhance multilingual support and add more languages.

Advanced Proctoring Features Develop AI-powered proctoring solutions to monitor exams and detect suspicious activities. Implement live proctoring options for real-time human invigilation.

User Interface Improvements Gather user feedback for continuous interface enhancements. Optimize the application for a seamless experience on mobile devices. Scalability and Performance Optimization Conduct extensive load testing to ensure the application can handle high user loads. Implement autoscaling features for efficient cloud resource management. Security Enhancements

Perform regular security audits to identify and mitigate vulnerabilities. Implement advanced encryption techniques to protect user data. Expanded Analytics and Reporting

Provide detailed analytics on exam performance and user behavior. Enable the generation of customizable reports tailored to specific needs. Integration and Collaboration

Integrate the application with popular Learning Management Systems (LMS). Develop APIs for seamless integration with third-party educational tools. Continuous Research and Development

Stay updated with the latest advancements in AI and machine learning. Conduct pilot programs with educational institutions to test new features and gather real-world feedback. Roadmap and Timeline Set short-term goals for immediate next steps and expected outcomes. Establish mid-term goals for planned features and improvements over the next few months.

Chapter 8 conclusion

The Arabic Automated Essay Scoring Using Transformer (LLMS) has shown promising results in accurately assessing the quality of written work in Arabic. The use of transformer-based models has allowed for more accurate and efficient scoring, as well as the ability to handle variations in language and writing style.

However, it is important to note that the effectiveness of the system may be limited by the quality and quantity of training data available. Additionally, the system may not be able to fully capture the nuances and complexities of Arabic language and culture, which could impact its accuracy and reliability.

Overall, the Arabic Automated Essay Scoring Using Transformer (LLMS) has the potential to revolutionize the way Arabic language is assessed and evaluated, and could have significant implications for education and research in the region. Further research and development is needed to fully realize its potential and address any limitations.

We have tried several models, such as Llama, T5, Bloom, and others, and we hope to improve them in the next step and obtain satisfactory results.

In the next step, we will improve the model to evaluate the validity of the answer and create a website that meets the project requirements and achieves the desire to improve this area.

References:

[1] LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS (arXiv:2106.09685, June 2021)

[2] Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks (arXiv:1908.10084, August 2019)

[3] A White Paper on Neural Network Quantization (arXiv:2106.08295, June 2021)

[4] Llama 2: Open Foundation and Fine-Tuned Chat Models (arXiv:2307.09288, July 2023)

[5] Attention Is All You Need (arXiv:1706.03762, June 2017)

[6] COMET: A Neural Framework for MT Evaluation (arXiv:2009.09025v2, September 2020)

[7] BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (arXiv:1910.13461v1, October 2019)

[8] LLaMA: Open and Efficient Foundation Language Models (arXiv:2302.13971v1, February 2023)

[9] Fine-Tuning Language Models from Human Preferences (arXiv:1909.08593v2, September 2019)

[10] Chain of Hindsight Aligns Language Models with Feedback (arXiv:2302.02676v8, February 2023)

[11] Scaling Instruction-Finetuned Language Models (arXiv:2210.11416v5, October 2022)

[12] Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (arXiv:1910.10683v4, October 2019)

[13] LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS (arXiv:2106.09685, June 2021)

[14] NLPOP: a Dataset for Popularity Prediction of Promoted NLP Research on Twitter (ACL Anthology: 2022)

[15] Mobile BERT: a Compact Task-Agnostic BERT for Resource-Limited Devices (arXiv:2004.02984v2, April 2020)

[16] Arabic-Centric Foundation and Instruction-Tuned Open Generative Large Language Models (arXiv:2308.16149v2, August 2023)

[17] ARBML: Democratizing Arabic Natural Language Processing Tools (ACL Anthology: 2020)

[18] An open access NLP dataset for Arabic dialects: Data collection, labeling, and model construction (arXiv:2102.11000v1, February 2021)

[19] AraBERT: Transformer-based Model for Arabic Language Understanding (arXiv:2003.00104v4, March 2020)

[20] AraGPT2: Pre-Trained Transformer for Arabic Language Generation (arXiv:2012.15520v2, December 2020)

[21] ARBERT & MARBERT: Deep Bidirectional Transformers for Arabic (arXiv:2101.01785v3, January 2021)

[22] Bidirectional Encoder Representations From Transformers (https://www.sciencedirect.com/topics/computer-science/bidirectional-encoder-representations-from-transformers, 2021)

[23] Evaluating Various Tokenizers for Arabic Text Classification (arXiv:2106.07540v2, June 2021)

[24] Sentiment Analysis of Arabic Algerian Dialect Using a Supervised Method (IEEE Xplore: 2019)

[25] AraNet: A Deep Learning Toolkit for Arabic Social Media (arXiv:1912.13072v2, December 2019)

[26] Arabisc: Context-Sensitive Neural Spelling Checker (ACL Anthology: 2020)

[27] Contextual Embeddings for Arabic-English Code-Switched Data (ACL Anthology: 2020)

[28] DAMO-NLP at SemEval-2023 Task 2: A Unified Retrieval-augmented System for Multilingual Named Entity Recognition (arXiv:2211.01786v2)

[29] UCAS-IIE-NLP at SemEval-2023 Task 12: Enhancing Generalization of Multilingual BERT for Low-resource Sentiment Analysis (arXiv:2005.11401v4, May 2020)

[30] Neural Framework for MT Evaluation (arXiv:2009.09025v2, September 2020)

[31] Human-Guided Fair Classification for Natural Language Processing (arXiv:2302.13971v1, February 2023)

[32] NLP Reproducibility For All: Understanding Experiences of Beginners (arXiv:1909.08593v2, September 2019)

[33] Natural Language Processing Methods to Identify Oncology Patients at High Risk for Acute Care with Clinical Notes (arXiv:2209.13860v2, September 2022)

[34] BLUEX: A benchmark based on Brazilian Leading Universities Entrance eXams (arXiv:2307.05410v1, July 2023)

[35] Visually-augmented pretrained language models for NLP tasks without images (arXiv:2212.07937v2, December 2022)

[36] Efficient NLP Model Finetuning via Multistage Data Filtering (arXiv:2207.14386v2, July 2022)

[37] Correcting Semantic Parses with Natural Language through Dynamic Schema Encoding (arXiv:2305.19974v1, May 2023)

[38] Retrieving data. Wait a few seconds and try to cut or copy again. (arXiv:2103.06678v2, March 2021)

[39] TCE at Qur'an QA 2022: Arabic Language Question Answering Over Holy Qur'an Using a Post-Processed Ensemble of BERT-based Models (arXiv:2206.01550v1, June 2022)

[40] Squeeze BERT: What can computer vision teach NLP about efficient neural networks? (arXiv:2006.11316v1, June 2020)

[41] Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP (arXiv:2212.14024v2, December 2022)

[42] Mixed-Precision Training for NLP and Speech Recognition with OpenSeq2Seq (arXiv:1805.10387v2, May 2018)

[43] Language Models are Few-Shot Learners (Language Models are Few-Shot Learners (neurips.cc), April 2020)

[44] Span Emo: Casting Multi-label Emotion Classification as Span-prediction (arXiv:2101.10038v1, January 2021)

[45] Pre-trained models for natural language processing (s11431-020-1647-3.pdf (springer.com), October 2020)

[46] Easy Transfer -- A Simple and Scalable Deep Transfer Learning Platform for NLP Applications (arXiv:2011.09463v3, November 2020)

[47] DTW at Qur'an QA 2022: Utilizing Transfer Learning with Transformers for Question Answering in a Low-resource Domain (ACL Anthology: 2022)

[48] NLPOP: a Dataset for Popularity Prediction of Promoted NLP Research on Twitter (ACL Anthology: 2022)

[49] Cross lingual Generalization through Multitask Finetuning (arXiv:2211.01786v2, November 2022)

[50] Evaluating Various Tokenizers for Arabic Text Classification (arXiv:2106.07540v2, June 2021)

Table of reference : Book 3.xlsx (sharepoint.com)