Faculty of Informatics Engineering

Department of Software Engineering

Large Language Models for Computer Vision

Computer vision course – project

Prepared by

Raghad Alhossny Mohamed al balkhi

Supervised by

Dr. Nisreen Sulayman

2023-2024

ABSTRACT

This report explores the integration of Large Language Models (LLMs) with Computer Vision (CV) for automatic image understanding and captioning. As LLMs advance natural language processing, their combination with CV offers new opportunities for enhanced image analysis. We cover fundamental concepts, from tokenization and embeddings to the evolution from RNNs and LSTMs to Transformers, with a focus on Vision Transformers (ViT) bridging image processing and language models.

The study compares a pipeline approach—integrating CV techniques like object detection, scene classification, pose estimation, and emotion recognition with LLMs for caption generation—with a direct approach using LLMs with visual understanding. The "Save Your Moments" app exemplifies these concepts, demonstrating how to automatically analyze and caption images for rich memory preservation.

Our findings highlight the efficacy of combining CV and LLMs, improving image understanding and generating contextually rich captions. We conclude by exploring the implications, potential applications, and future research in multimodal AI, emphasizing the synergy between language models and computer vision.

ملخص

يستعرض هذا التقرير التكامل بين النماذج اللغوية الكبيرة (LLMs) وتقنيات الرؤية الحاسوبية (CV) لتحليل الصور ووصفها تلقائيًا. مع تطور LLMs في معالجة اللغة الطبيعية، يوفر دمجها مع CV فرصًا جديدة لتحليل الصور بعمق.

نغطي المفاهيم الأساسية مثل التجزئة (Tokenization) والتمثيلات الشعاعية (Embeddings)، والتطور من الشبكات العصبية التقليدية Recurrent Neural) (Networksإلى المحولات(Transformers) ، مع التركيز على محولات الرؤية Vision) .(Transformersنناقش طريقتين: النهج المتسلسل (Pipeline) الذي يدمج تقنيات مثل كشف الأشياء (Object Detection) وتصنيف المشاهد (Scene Classification) وتقدير الوضعيات (Pose Estimation) والتعرف على المشاعر Emotion) (Recognition، والنهج المباشر باستخدام LLMs مع قدرات الرؤية.

تطبيق "Save Your Moments" يعرض هذه المفاهيم عمليًا، ويظهر كيف يمكن تحليل الصور ووصفها للحفاظ على الذكريات. النتائج تؤكد فعالية الجمع بين CV و LLMsفي تحسين فهم الصور وتوليد أوصاف غنية بالسياق. نختتم بنظرة على التطبيقات المستقبلية والبحث في الذكاء الاصطناعي متعدد.

TABLE OF CONTENT

Contents

List of Figures:

List of tables:

List of abbreviations

Table 1 list of abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
LLMs	Large Language Models
NN	Nural Networks
GPT	Generative Pre-trained Transformer
LLAMA	Large Language Model Meta AI
BERT	Bidirectional Encoder Representations from Transformers
LORA	low-rank-adaptation
BPE	Byte Pair Encoding
C2C	Computer to computer

Chapter1 Introduction to large language models

1. Introduction:

In this chapter, we introduce large language models (LLMs) as a transformative trend in artificial intelligence, defining their role and highlighting their widespread usage in daily applications. We also cover key terminologies associated with LLMs to provide a foundational understanding of this cutting-edge technology.

2. Large Language Models: Defining the Next Era of Technology

Artificial intelligence (AI) has radically transformed our lives and businesses in the past decade, with large language models (LLMs) at the epicenter of this shift. In essence, LLMs are advanced deep learning models that change the way we interact with devices, websites, and information in general.

Looking back over the past five years, LLMs consistently ranked among the top 12 trends.

LLMs have become a focal point in the AI landscape, especially after the launch of **ChatGPT** in **November 2022**. The subsequent year witnessed a surge in the development of sophisticated multimodal models and a vibrant open-source landscape for AI

LLMs are now among the top 14% of all emerging technologies, indicating their potential to create new job roles, skill requirements, and disrupt existing jobs.

Key Takeaways covered by TrendFeedr about LLM as a trend:

TrendFeedr identifies future industry and tech trends via advanced, proprietary algorithms. With a focus on trend discovery, clustering, and

analysis, the AI-powered platform screens thousands of trends each month to track their development and curate actionable insights.

- LLMs rank in the top 14% of all 20K+ trends covered by TrendFeedr, with an annual growth rate of over 90%, a trend magnitude of 85.05%, and a trend maturity of 24.5%.
- Approximately, 562 organizations are engaged with LLMs, with a total funding of $18.2 billion.
- LLM companies are also involved in other emerging trends like Neural Machine Translation, Voice Intelligence, Transfer Learning, Machine Learning as a Service, and Automated Speech Recognition.
- The United States, India, China, Canada, and Germany take the lead in LLM adoption.
- Media exposure for LLMs increased 30-fold in 2023 alone.

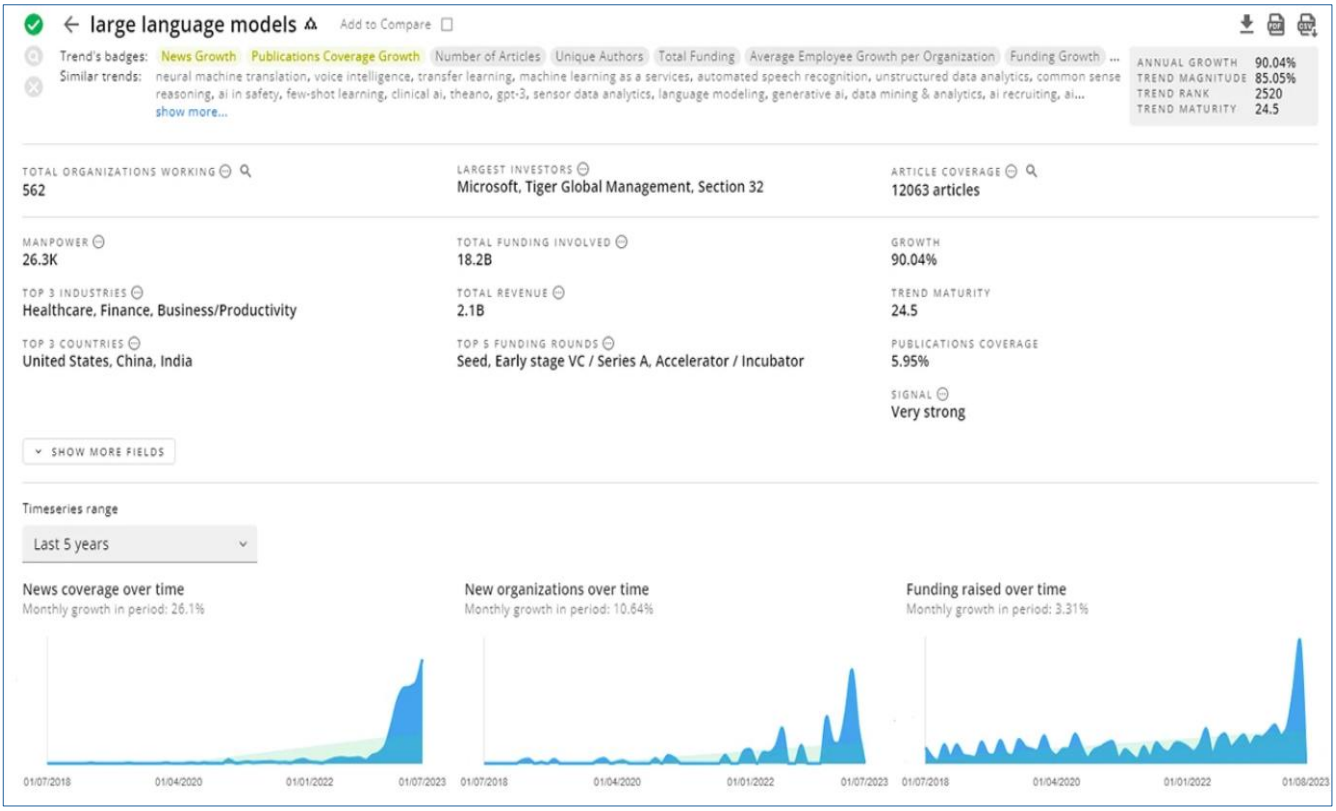


Figure 1 TrendFeedr's Trend Card

The global distribution of the 562 companies operating in LLMs:

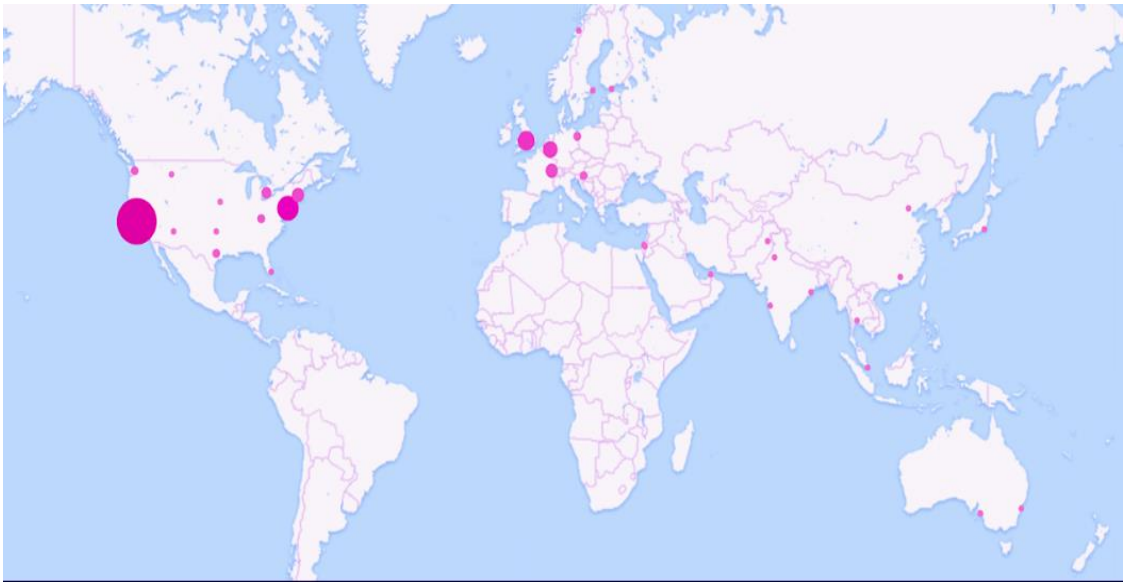


Figure 2 global distribution of the companies operating in LLMs

3. What Is a Large Language Model:

A large language model (LLM) is a type of artificial intelligence (AI) program that can recognize and generate text, among other NLP tasks. LLMs are trained on huge sets of data — hence the name "large." LLMs are built on machine learning: specifically, a type of neural network called a transformer model.

In simpler terms, an LLM is a computer program that has been fed enough examples to be able to recognize and interpret human language or other types of complex data.

Many LLMs are trained on data that has been gathered from the Internet — thousands or millions of gigabytes' worth of text. But the quality of the samples

impacts how well LLMs will learn natural language, so an LLM's programmers may use a more curated data set.

LLMs use a type of machine learning called deep learning in order to understand how characters, words, and sentences function together. Deep learning involves the probabilistic analysis of unstructured data, which eventually enables the deep learning model to recognize distinctions between pieces of content without human intervention.

LLMs are then further trained via tuning: they are fine-tuned or prompt-tuned to the particular task that the programmer wants them to do, such as interpreting questions and generating responses, or translating text from one language to another.

A key characteristic of LLMs is their ability to respond to unpredictable queries. A traditional computer program receives commands in its accepted syntax, or from a certain set of inputs from the user.

By contrast, an LLM can respond to natural human language and use data analysis to answer an unstructured question or prompt in a way that makes sense. Whereas a typical computer program would not recognize a prompt like "What are the four greatest funk bands in history?", an LLM might reply with a list of four such bands, and a reasonably cogent defense of why they are the best. In terms of the information they provide, however, LLMs can only be as reliable as the data they ingest. If fed false information, they will give false information in response to user queries.

4. The Rise of LLMs in Daily Applications:

As technology continues to improve every single day, LLM use cases are also becoming more sophisticated and diverse.

Large language models' ability to generate text in real-time has made them invaluable in enhancing search engines, powering virtual assistants, and improving language translation services. While these are just three examples, there are many more LLM use cases.

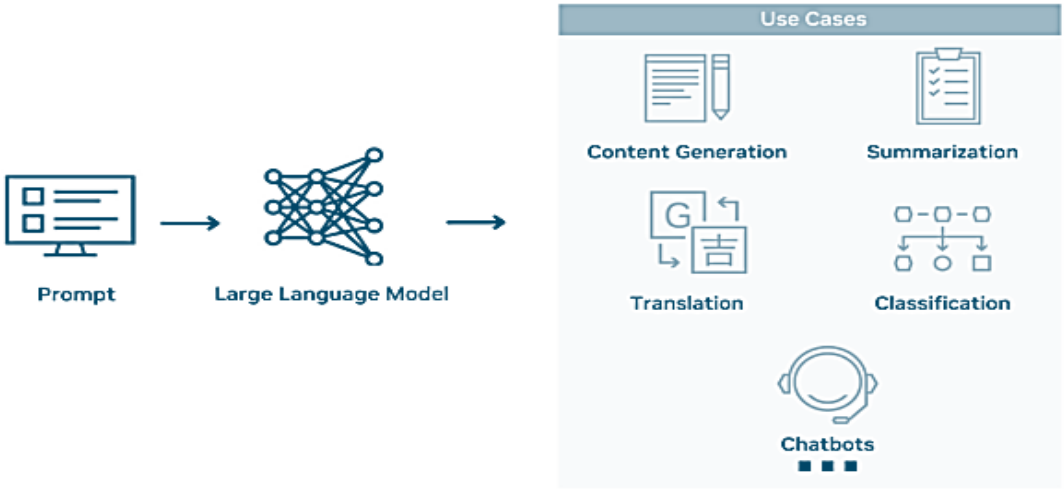


Figure 3 LLM use case

Here we will mention a few examples of main use cases of LLM:

- **Content generation:** LLM applications are especially good at content generation. They can be used to automatically create texts for various purposes, including articles, blog posts, marketing copy, video scripts, and social media updates. Moreover, LLM-backed generative AI apps can adapt to different writing styles and tones, making them versatile for generating content that resonates with specific target audiences.

  Businesses and content creators harness these models to streamline content production, saving time and effort in the writing process.

- **Search and recommendation:** LLMs are capable of understanding and processing natural language queries with unprecedented accuracy and context. When integrated into search engines, these models can interpret the intent behind a user's query and deliver more relevant and precise results. They can also generate summaries of content, making it easier for users to find the information they need quickly.

  In recommendation systems, LLMs analyze user preferences, search history, and interaction data to personalize content suggestions. They can predict user needs, thereby enhancing the entire user experience.

- **Code development:** Large language models can assist programmers in writing, reviewing, and debugging code. These models can understand and generate code snippets, suggest completions, and even write entire functions based on brief descriptions. For instance, a developer might input a comment like "sort a list of numbers in ascending order," and the LLM can provide the corresponding code.

  Furthermore, LLMs can translate code between different programming languages, making it easier for developers to work with unfamiliar syntax or migrate projects to a new language

- **Education:** LLM applications are increasingly being used in education to personalize learning and provide tutoring.

  LLMs can adapt to individual student's learning styles and pace, offering customized explanations and feedback. For instance, a model can generate interactive reading materials that adjust complexity based on the student's comprehension level or provide real-time language translation to aid foreign students. Just like being virtual tutors, LLMs can answer students' questions, guide them through problem-solving steps, and even motivate them with encouraging messages.

5. Key Vocabulary in the Context of LLMs:

Understanding large language models (LLMs) requires familiarity with specific terminology central to their development and functionality.

5.1 Parameters:

In a LLM that realis on NN, Training parameters in LLMs refer to the variables that the model learns from the data during the training phase. These include *weights and biases*, which the model adjusts through backpropagation – a process that minimizes the difference between the model's prediction and the actual output.

- **Weights:** It is a key trainable parameter in a neural network to adjust the influence of each input on the final output.

  For instance, if we're trying to predict house prices based on size and location, these two factors become the inputs to our model. However, they may not equally influence the price. The size of the house, often, has a more significant impact on the price than its location. This influence is determined by the "weights" in our model. If size has a larger weight, it means it has a greater contribution to the final price prediction. These weights are learned and refined during the model's training phase.

- **Biases:** It is another key trainable parameter that acts like an adjustable constant, providing flexibility to the model's predictions.

> Think of bias as a kind of baseline or starting point for predictions.

For instance, houses in a specific city might generally start at a certain price, regardless of size or location. This baseline price can be thought of as the bias in our model. Even when the size or location isn't known, the model can still make an accurate prediction thanks to this bias. So, biases help the model make more flexible and precise predictions under various conditions.

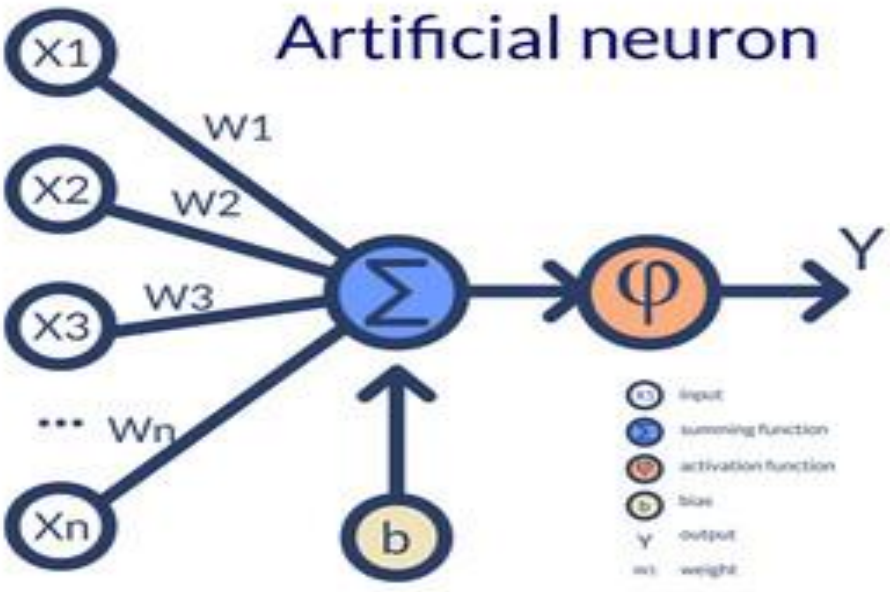


Figure 4 Artificial neuron

The quality of a language model is directly influenced by the tuning and optimization of its training parameters.

a. **Prediction Accuracy**: Optimal training parameters lead to higher prediction accuracy. They help the model generalize from the training

data to unseen data, thereby improving its performance on various NLP tasks, like text generation, translation, and summarization.

b. **Overfitting and Underfitting**: The right balance of parameters can help prevent underfitting (where the model is too simple to capture useful patterns) and overfitting (where the model memorizes the training data, leading to poor performance on new data).
c. **Interpretability**: Although LLMs are often seen as "black boxes", understanding their training parameters can provide some insight into their decision-making process. For instance, looking at the weights of different connections can help understand which features the model deems most important.
d. **Bias and Fairness**: Training parameters can inadvertently lead to biased predictions if the training data contains biased patterns. Careful monitoring and adjustment of these parameters can help mitigate such biases, leading to more fair and responsible AI.

Determining the appropriate size of the training parameters— the model's complexity—is a critical decision in designing LLMs. Here are a few key considerations:

a. **Data Availability**: As a rule of thumb, the more high-quality data available for training, the more complex the model can be.

b. **Computational Resources**: Training larger models requires more computational resources, both in terms of processing power and memory. This must be balanced against the potential benefits of a more complex model.

c. **Performance Requirements**: The complexity of the model should match the complexity of the task. For instance, simple tasks may not

benefit from very complex models, while complex tasks may require larger models.

d. **Trade-off Between Precision and Speed**: Larger models may provide better performance but at the cost of slower prediction speeds. Depending on the application, this trade-off may or may not be acceptable.

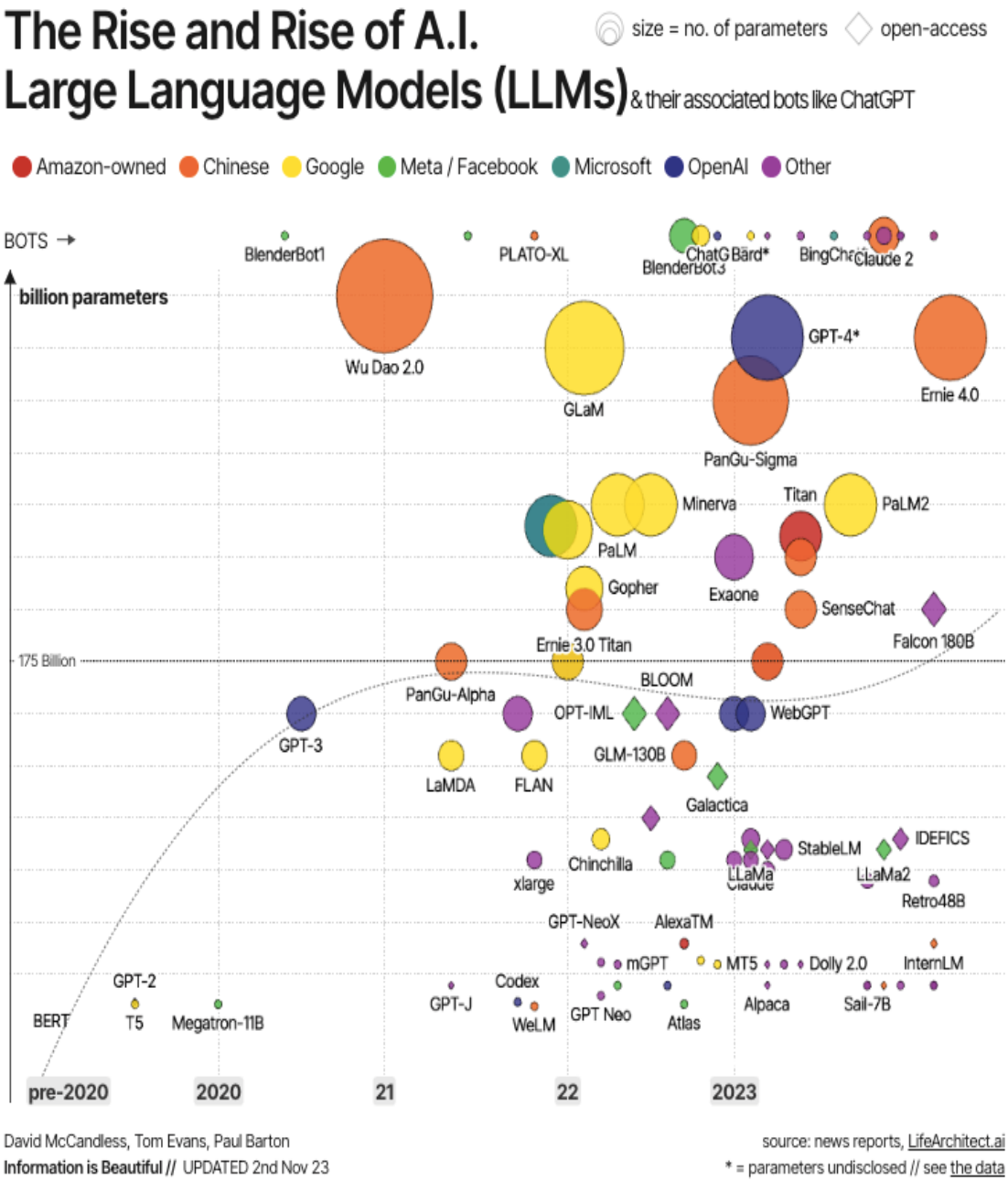


Figure 5 the parameter counts over time for different models

5.2 Tokens:

The Building Blocks of LLMs

Tokens are the basic units of data processed by LLMs. In the context of text, a token can be a word, part of a word (subword), or even a character — depending on the tokenization process.

When text is passed through a tokenizer, it encodes the input based on a specific scheme and emits specialized vectors that can be understood by the LLM. The encoding scheme is highly dependent on the LLM. The tokenizer may decide to convert each word and a part of the word into a vector, which is based on the encoding. When a token is passed through a decoder, it can be easily translated into text again.

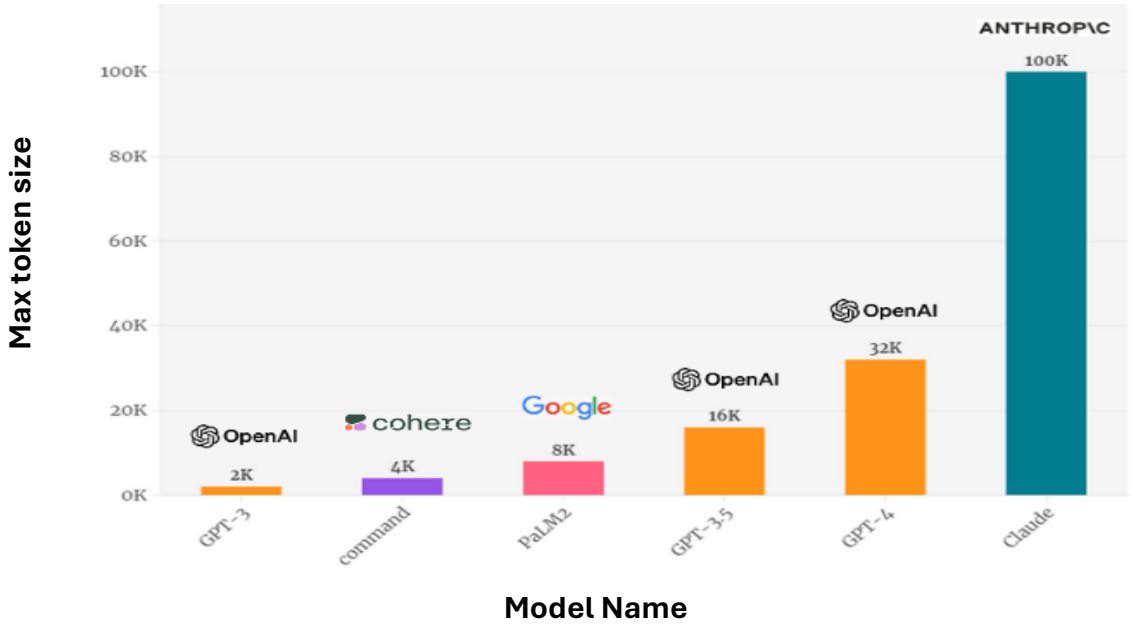


Figure 6 Max token size for different LLM models

It's common to compare large language models (LLMs) by the number of tokens they were trained on.

It is a common practice because the amount of training data directly influences the model's performance and capabilities.

The number of tokens a model can handle at once, known as the 'token limit', is a crucial metric. This limit impacts both the length of text the model can consider and the amount of context it can use when generating responses.

Understanding token limits can help you optimize your usage of these models. For example, when you're using a model to generate text, if your input prompt is too long, the model might not have enough tokens left to provide a meaningful response. Conversely, a very short prompt might not give the model enough context to generate a useful reply.

5.3 The cost of building LLM:

Building a substantial language model from scratch is often unnecessary for many LLM applications. Techniques like **prompt engineering** or **fine-tuning existing models** are generally more effective than the laborious task of starting from scratch.

The financial considerations in question revolve around the computational expenses, using Llama 2 as a benchmark — a recently developed large language model by Meta. The computational costs pertain to both the **7 billion parameter** and **70 billion parameter** versions of the model. For the 7 billion parameter model, the training required approximately **180,000 GPU hours,** while the 70 billion

parameter model, being ten times larger, necessitated **1.7 million GPU hours.** Employing a physicist's approach, we can generalize based on these Llama 2 figures. A 10 billion parameter model is estimated to require around 100,000 GPU hours, while a 100 billion parameter model might demand roughly a million GPU hours for training.

Two alternatives present themselves. Firstly, one can opt to rent GPUs and compute power from major cloud service providers like Nvidia A100, incurring costs ranging from $1 to $2 per GPU hour. By simple multiplication, the estimated training cost for a 10 billion parameter model is approximately $50,000, and for a 100 billion parameter model, it could reach about $1.5 million.

Alternatively, one can choose to invest in purchasing the hardware. Considering an A100 costs around $110,000, forming a GPU cluster with about 1,000 GPUs would lead to hardware costs in the ballpark of $10 million. However, this is not the sole expense; running such a cluster for weeks consumes substantial energy. If training a 100 billion parameter model consumes about 1,000 megawatt hours at a rate of $100 per megawatt hour, the total marginal cost for training would be roughly $100,000.

5.4 Prompt engineering:

The process of crafting, refining, and optimizing the input prompts given to an LLM in order to achieve desired outputs. Prompt engineering plays a crucial role in determining the performance and behavior of models like those based on the GPT architecture.

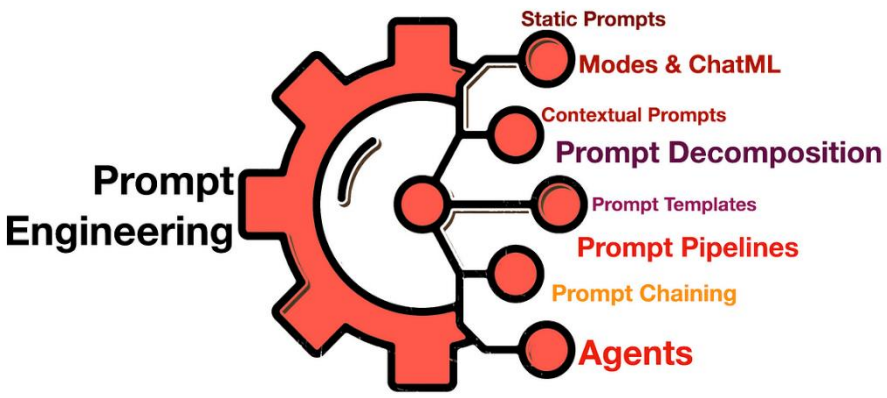


Figure 7 prompt engineering

Given the vast knowledge and diverse potential responses a model can generate, the way a question or instruction is phrased can lead to significantly different results. Some specific techniques in prompt engineering include:

- **Rephrasing:** Sometimes, rewording a prompt can lead to better results. For instance, instead of asking "What is the capital of France?" one might ask, "Can you name the city that serves as the capital of France?"
- **Specifying Format:** For tasks where the format of the answer matters, you can specify it in the prompt. For example, "Provide an answer in bullet points" or "Write a three-sentence summary."

- **Leading Information:** Including additional context or leading information can help in narrowing down the desired response. E.g., "Considering the economic implications, explain the impact of inflation."

Prompt engineering is important in the context of retrieval-augmented-generation as well. To get the best results, a RAG prompt needs to specify a set of instructions to the LLM so that it considers the provided facts in the right way, and can provide the correct response.

5.5 Fine-tune:

Fine-tuning is one way to perform transfer learning. With fine-tuning, you take a pre-trained model and continue training it on a specific (usually smaller, more limited) dataset.

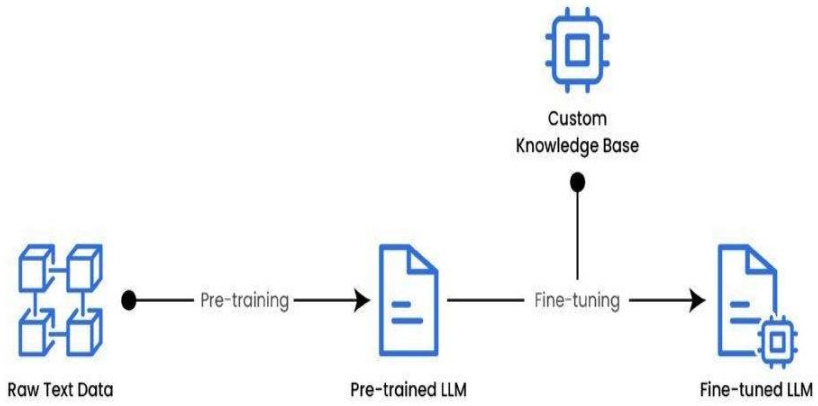


Figure 8 fine-tune process

There are a few varieties that are common ways to perform fine-tuning, for example:

- Continue training of complete NN on a specific dataset
- Freeze some layers and continue training the other layers
- Add a new layer for a new type of task and only train that layer

Regardless of which specific technique you use for fine-tuning, it's a much less complicated and significantly less expensive task than full training of an LLM.

Recently techniques like LORA (low-rank-adaptation) were suggested for fine-tuning which make it even faster and less expensive, while keeping the performance of fine-tuning similar.

6. How do large language models integrate with computer vision:

Picture a world where machines can not only see but also describe what they see in a way that is insightful and relatable to humans. This is the world we are stepping into, thanks to the confluence of two of the most groundbreaking technologies in artificial intelligence: Large Language Models (LLMs) and Computer Vision.

Over the years, computer vision has empowered machines to comprehend images and videos, facilitating capabilities like object detection, image classification, pattern recognition, and situational analysis. At the same time, large language models have allowed machines to understand and generate

human-like language. These two areas are beginning to intersect, holding immense potential for enterprises across industries.

While computer vision is already revolutionizing many industries, integrating it with large language models can take its capabilities several notches higher. The goal is to teach these machines to see and generate human-like language and respond to textual prompts. As a result, providing more detailed insights about the visuals and video streams.

Integrating large language models with computer vision allows operators to query, using text prompts, an infinite number of video streams at the same time with natural language, enhancing computer-to-computer (C2C) interactions.

Until now, AI solutions have largely been segregated based on their computational power, use case needs, algorithm designs, and data type requirements for model training. However, the demand for multi-modal solutions that deliver targeted business value and address as many adjacent needs as possible is rising. Integrating large language models and computer vision is a step in this direction, bringing us closer to realizing the dream of a highly competent digital assistant.

The integration of large language models and computer vision is heralding the advent of next-gen AI technology, where machines are trained to see and tell us what they see. For organizations, the convergence of these technologies facilitates the classification of enterprise data, generates prompts for specific visual content, and provides customized insights for actionable decision-making.

Chapter2 Theoretical Study

1. Introduction:

In this chapter, we explore the theoretical underpinnings of large language models, beginning with the evolution from traditional neural networks to the groundbreaking Transformer architecture, as epitomized by the seminal paper "Attention is All You Need." We will delve into the fundamental concepts and architectures that form the backbone of modern NLP systems, with a particular focus on the key components of the Transformer architecture. This includes an in-depth look at attention mechanisms, positional encoding, and other critical innovations that enable the powerful performance of Transformers in understanding and generating human language.

2. Fundamental Concepts and Architectures

2.1 From Neural Networks to Transformers

2.1.1 Recurrent Neural Networks (RNNs) and Long Short-term Memory (LSTM)

Recurrent Neural Networks (RNNs):

- Process sequential data by maintaining an internal state (memory)
- At each time step, RNN takes input X(t) and previous state, outputs Y(t) and updates state
- Allows handling variable-length sequences
- Useful for tasks like language modeling, speech recognition, and time series prediction

Key limitation of basic RNNs:

- Struggle with long-term dependencies due to vanishing/exploding gradient problem
- Can't work in parallel due to dependency on previous state, thus requires many time steps.

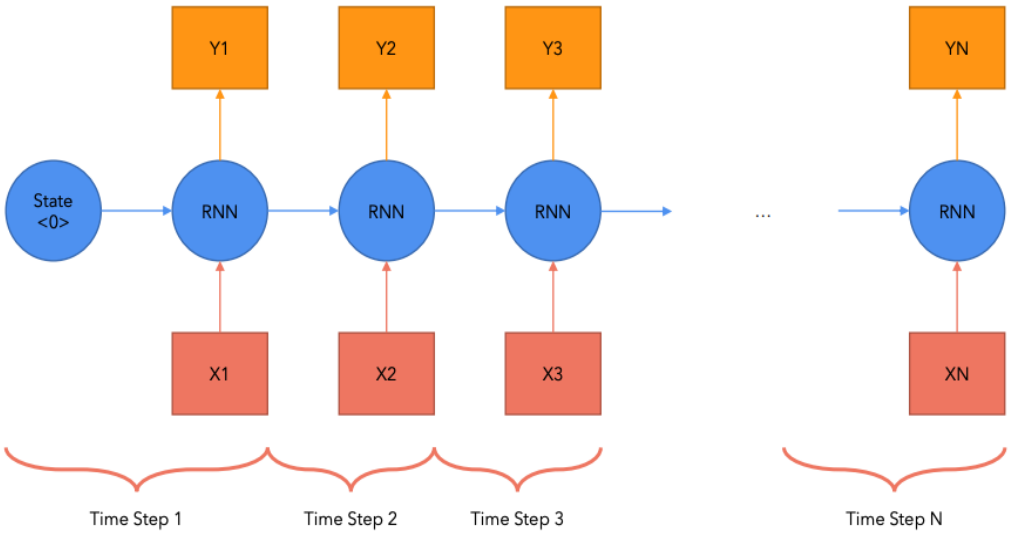


Figure 9 RNN Time Steps

Long Short-Term Memory (LSTM):

- Special type of RNN designed to address long-term dependency issue
- Introduces gating mechanisms: input gate, forget gate, and output gate
- Gates control information flow, allowing network to selectively remember or forget information
- More effective at capturing long-range dependencies in sequences

Advantages of LSTMs:

- Better at preserving information over many times steps.
- Mitigates vanishing gradient problem.
- Improved performance on tasks requiring long-term memory.

Limitations leading to Transformers:

- Sequential processing limits parallelization
- Still face challenges with very long sequences
- Computationally expensive for long sequences

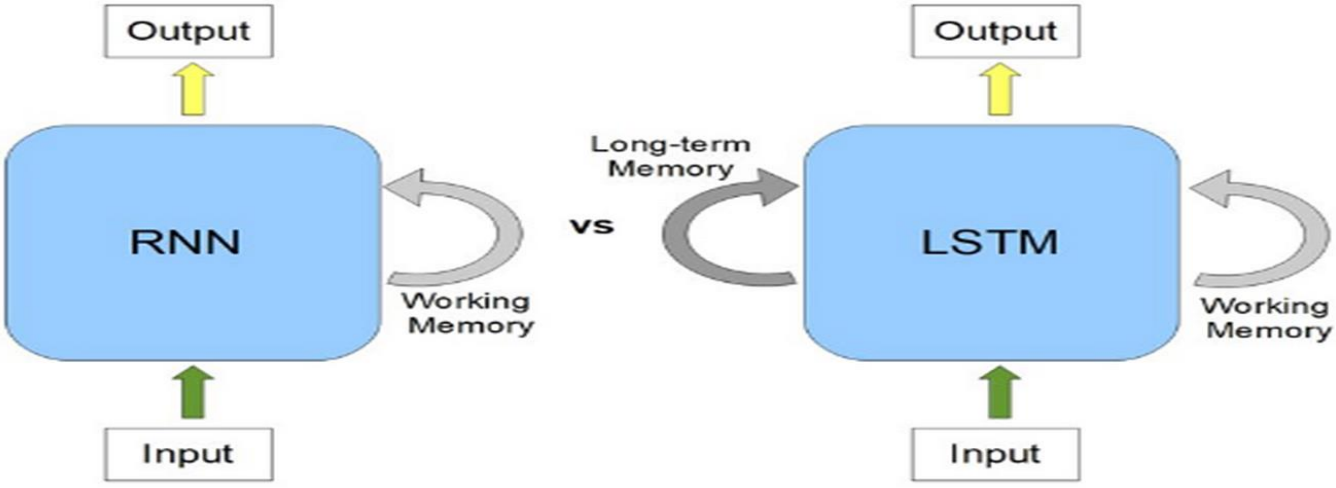


Figure 10 LSTM Evolution over RNN

2.1.2 The Transformer Architecture: "Attention is All You Need"

The Transformer architecture, introduced in the **2017** paper **"Attention is All You Need" by Vaswani et al.,** revolutionized natural language processing and sequence modeling tasks. Key points:

- Eliminated recurrence and convolutions used in previous sequence transduction models
- Relied entirely on attention mechanisms to draw global dependencies between input and output
- Enabled significant parallelization, leading to faster training on more data
- Achieved state-of-the-art performance on various NLP tasks
- Became the foundation for modern language models like BERT, GPT, and their successors
- Encoder – Decoder as shown in the figure below, where the **left** part is called **Encoder**, while the **right** part is called **Decoder**.

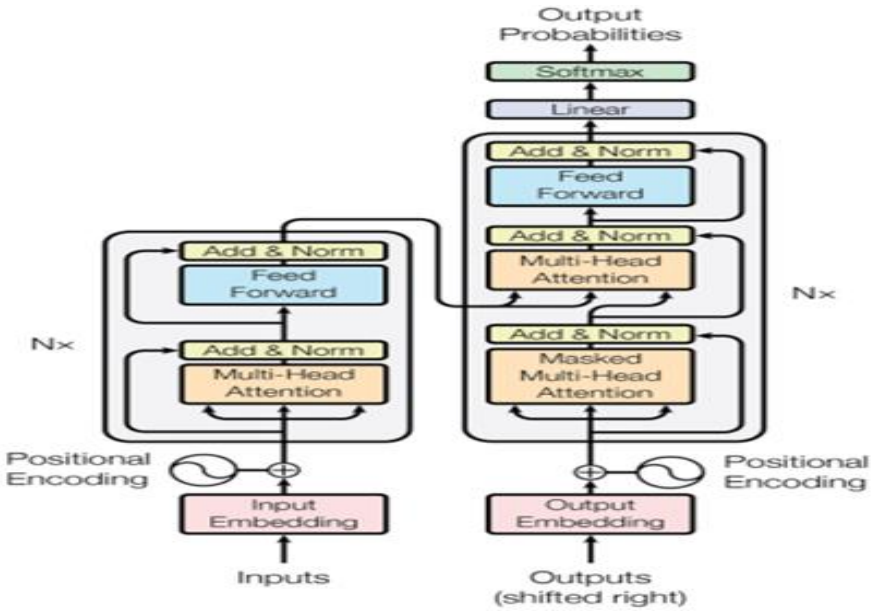


Figure 11 Transformer Architecture

2.1.3 Positional Encoding and Self-Attention Mechanisms

Positional Encoding:

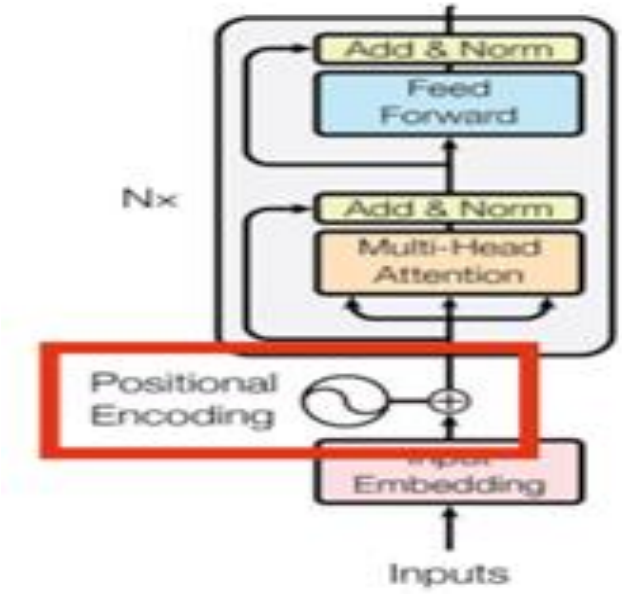


Figure 12 positional encoding

- Purpose: Inject information about the relative or absolute position of tokens in the sequence
- Formula: For position **pos** and dimension **i**:

$$PE(pos, 2i) = sin(pos / 10000\text{^}(2i/d\_model))$$

$$PE(pos, 2i + 1) = cos(pos / 10000\text{^}(2i/d\_model))$$

- Added to the input embeddings before feeding into the encoder/decoder stacks
- Allows the model to make use of the order of the sequence

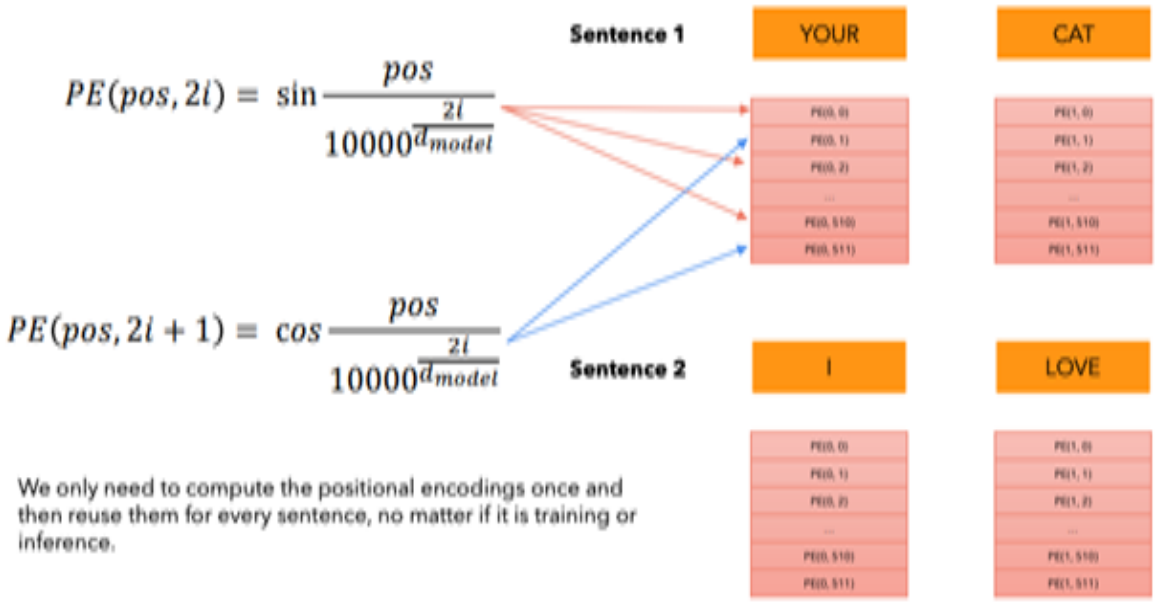


Figure 13 positional encoding calculation

Self-Attention Mechanism:

- Core component of the Transformer, allowing it to weigh the importance of different words in a sequence
- Steps:
- Create Query (Q), Key (K), and Value (V) matrices from input
- Calculate attention scores:

$$score = softmax(QK^T / \sqrt{d\_k})$$

- Apply scores to values:

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d\_k})V$$

- Multi-head attention: Allows the model to jointly attend to information from different representation subspaces

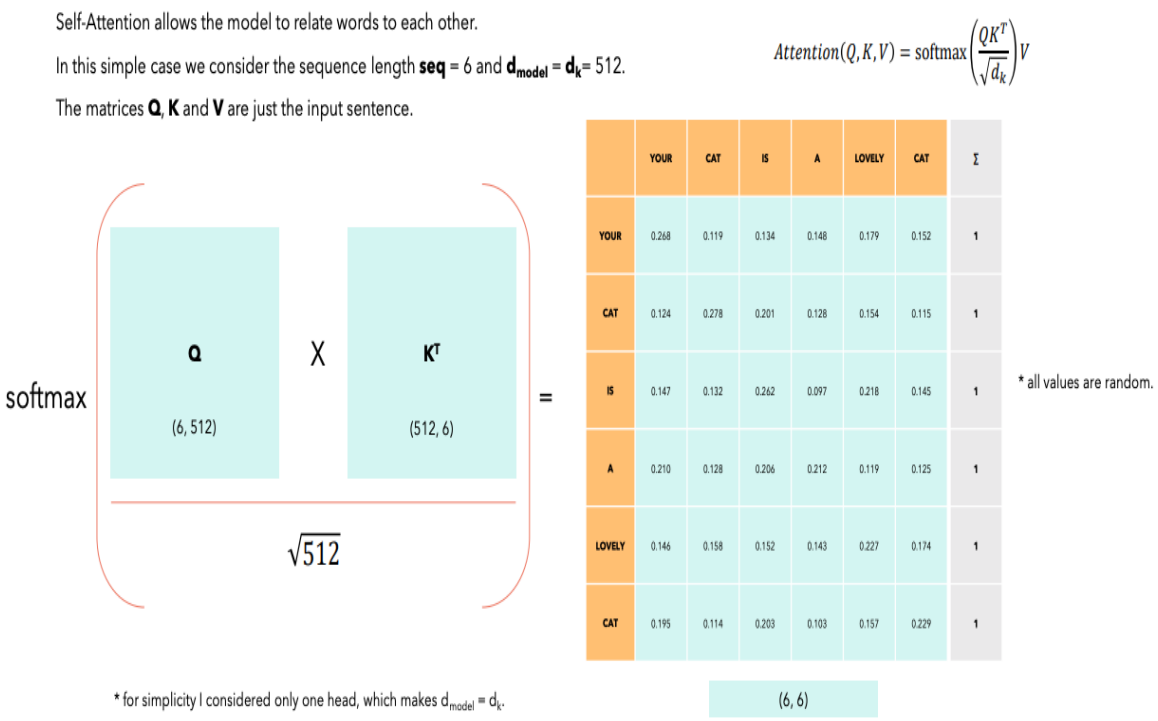


Figure 14 self-attention calculation

Query, Keys, and Values in Self-Attention:

The self-attention mechanism relies on three main components: Queries, Keys, and Values. These are derived from the input sequence through linear transformations.

1. Query (Q): Represents the current word's "question" to other words in the sequence.
2. Key (K): Represents how relevant each word in the sequence is to answering the query.
3. Value (V): Carries the actual content of the words to be aggregated.

Process:

1. For each word, create a Query vector and compare it with all Key vectors.
2. The dot product between Query and Key determines the attention score.
3. Attention scores are normalized using softmax.
4. The final output is a weighted sum of Value vectors, where weights are the attention scores.

Formula:

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d_k})V$$

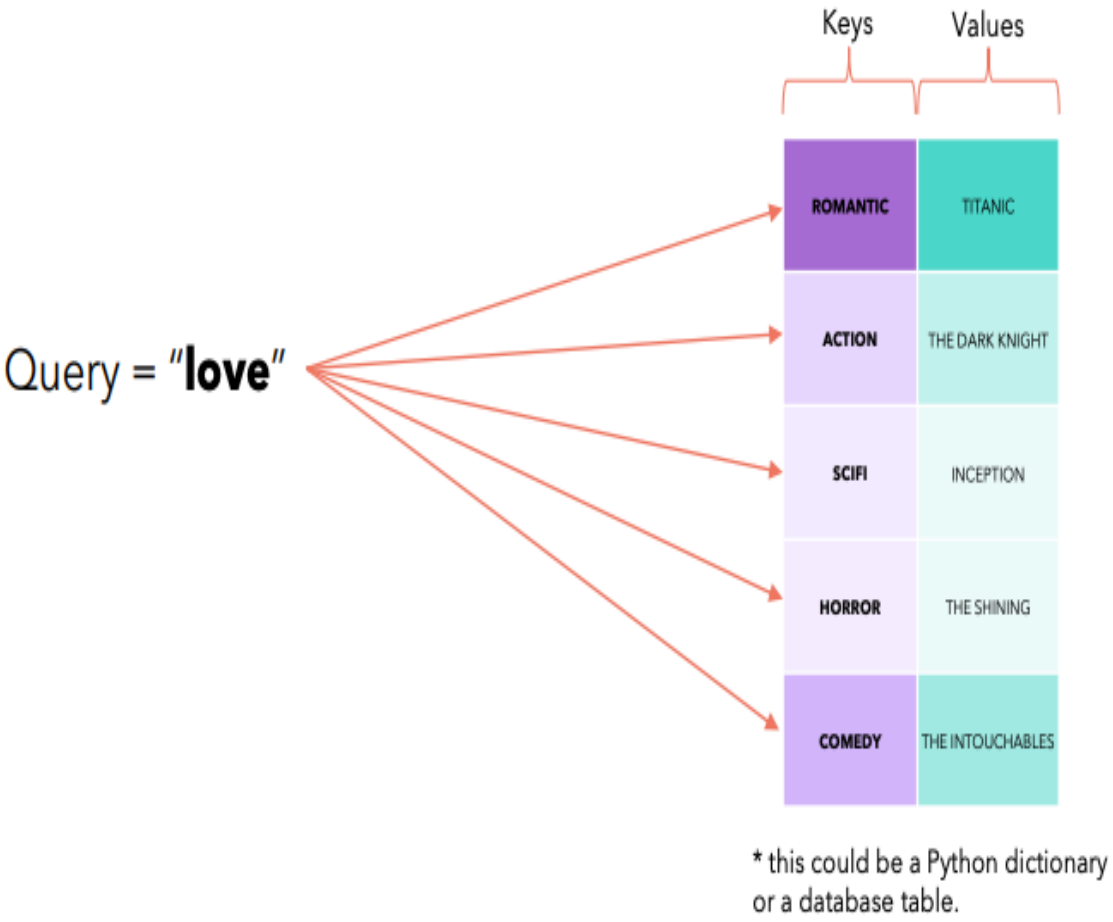


Figure 15 query key and value

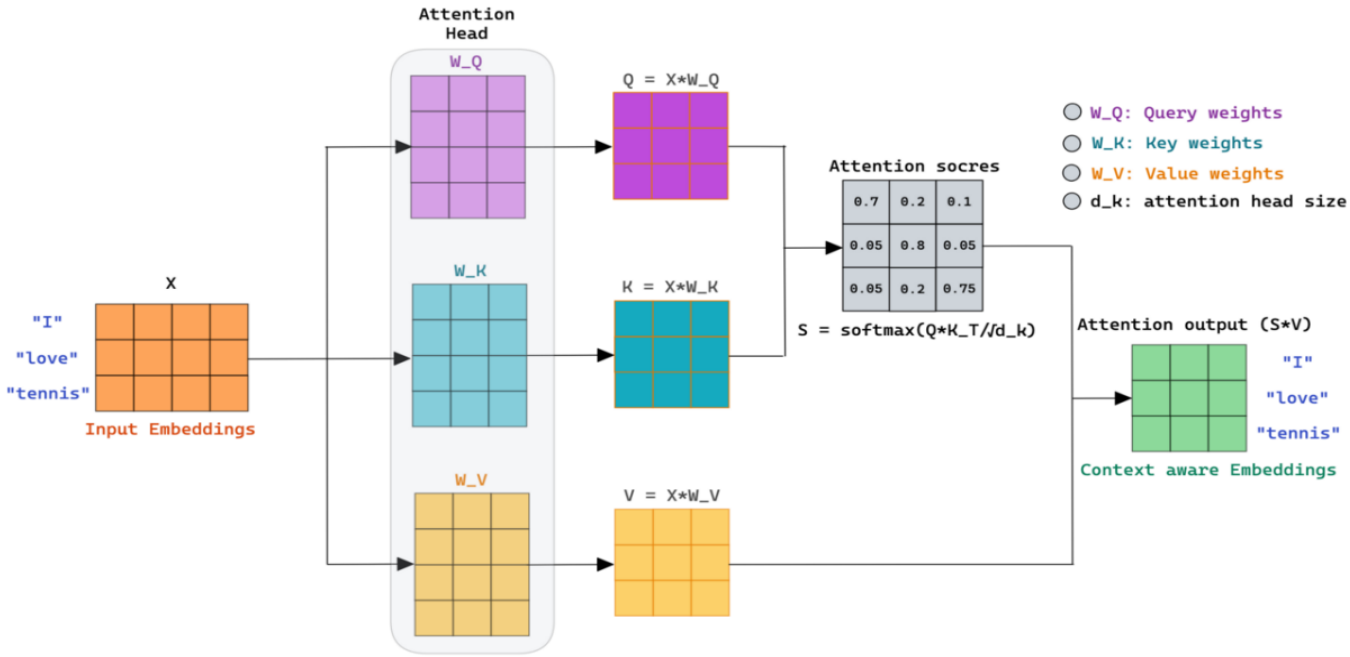


Figure 16 Self Attention full process

This figure illustrates the self-attention mechanism in action. Input embeddings **(X)** for the words "I", "love", and "tennis" are transformed into Query **(Q),** Key **(K),** and Value **(V)** matrices using learned weight matrices **(W_Q, W_K, W_V).** Attention scores are computed using **Q** and **K**, then applied to V to produce context-aware embeddings. The **d_k** parameter represents the dimension of the attention head. This process allows each word to attend to all others, capturing complex relationships within the sequence.

Key advantages of Transformers with Attention & Positional Encoding:

- Captures **long-range dependencies** more effectively than RNNs

- Allows **parallel computation**, significantly speeding up training

- Provides **interpretable attention weights**, offering insights into model decisions

2.2 Transformer Architecture Components

2.2.1 Encoder-Decoder Structure

The Transformer architecture consists of an encoder and a decoder.

Encoder:

- Processes the input sequence
- Each layer has two sub-layers: multi-head self-attention and feed-forward neural network
- Uses residual connections and layer normalization

Decoder:

- Generates the output sequence
- Each layer has three sub-layers: masked multi-head self-attention, multi-head attention over encoder output, and feed-forward network
- Also uses residual connections and layer normalization

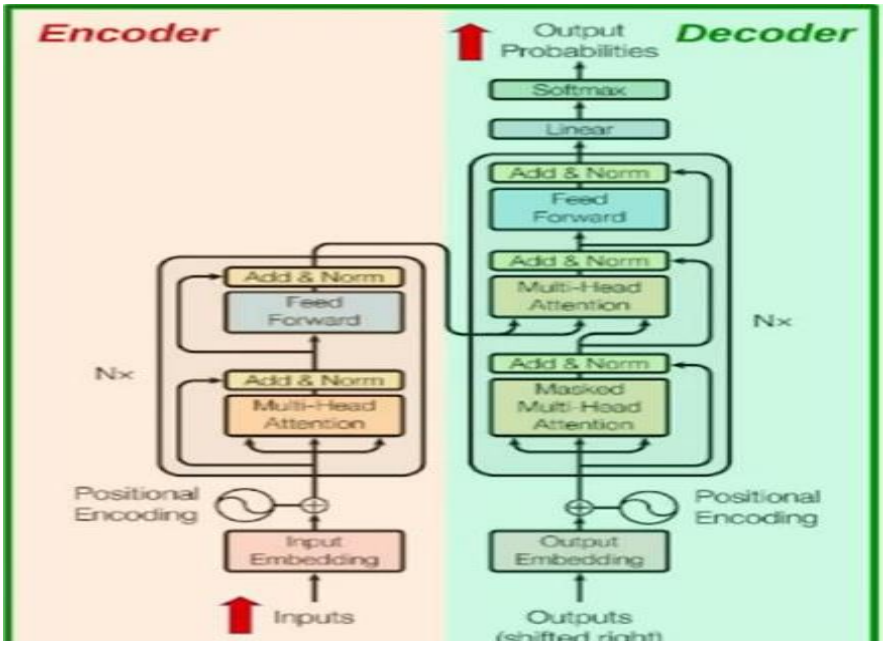


Figure 17 Encoder-Decoder Architecture

2.2.2 Multi-Head Attention

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

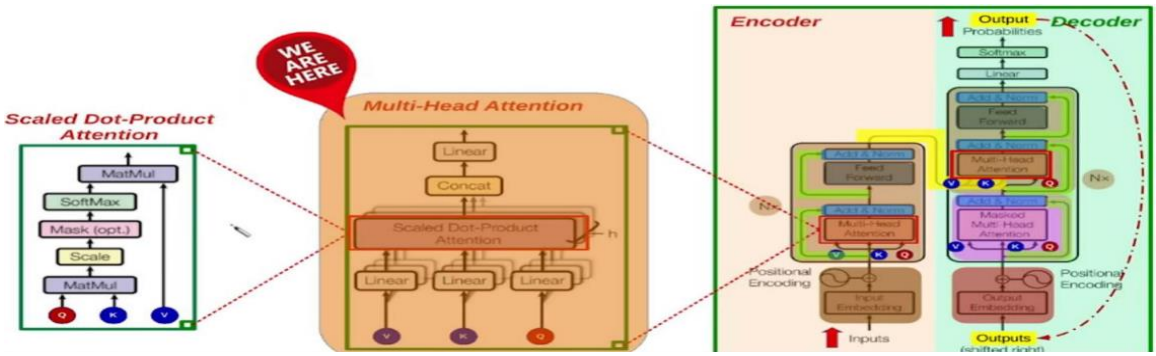


Figure 18 multi-head attention

- Consists of several attention layers running in **parallel**

- Each head uses different learned projections for Q, K, and V

- Outputs are concatenated and once again projected

Formula:

$$MultiHead(Q,K,V) = Concat(head1,\ldots,headh)W\hat{}O$$

where $headi = Attention(QW\hat{}Q\_i, KW\hat{}K\_i, VW\hat{}V\_i)$

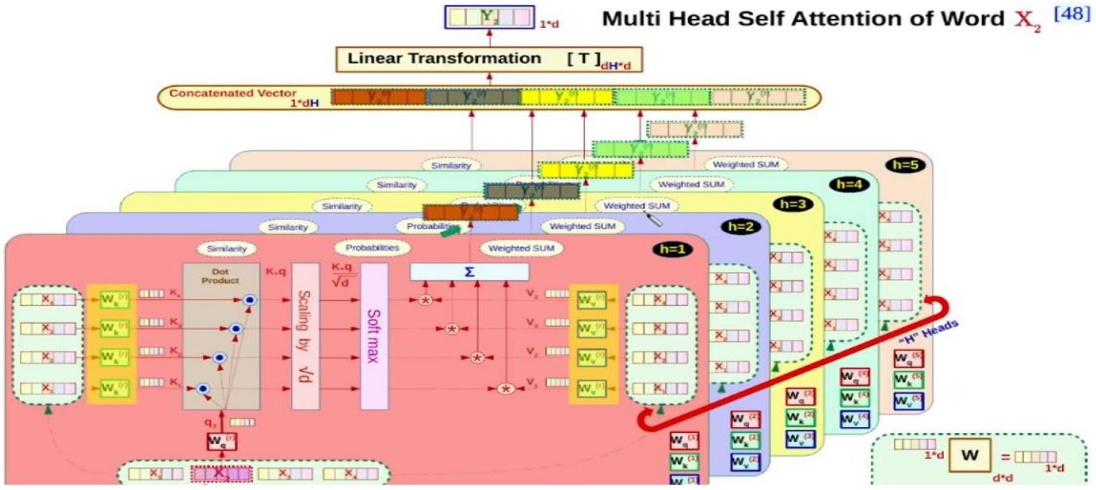


Figure 19 Multi-Head Attention Parallelism

2.2.3 Feed-Forward Networks

In transformer models, each encoder and decoder layer include a fully connected feed-forward network. These networks play a crucial role in processing the information attended by the model which helps in learning complex patterns.

Characteristics:

- Applied to each position separately and identically.

- Consists of two linear transformations with a ReLU activation in between.

Mathematical Formula:

$$FFN(x) = max(0, xW1 + b1)W2 + b2$$

where:

- (x) is the input.

- ($W1$) and ($W2$) are weight matrices.

- ($b1$) and ($b2$) are bias vectors.

Role in Transformers:

The feed-forward network processes the output of the multi-head attention mechanism, introducing non-linearity and enabling the model to capture complex relationships in the data. This is critical for the model's performance, especially in handling natural language tasks and vision-related tasks in LLMs.

Example:

Consider a sentence encoded by a transformer. After the attention mechanism determines the relevance of each word to others, the feed-forward network transforms these weighted sums, adding non-linearity and allowing the model to learn intricate dependencies and representations.

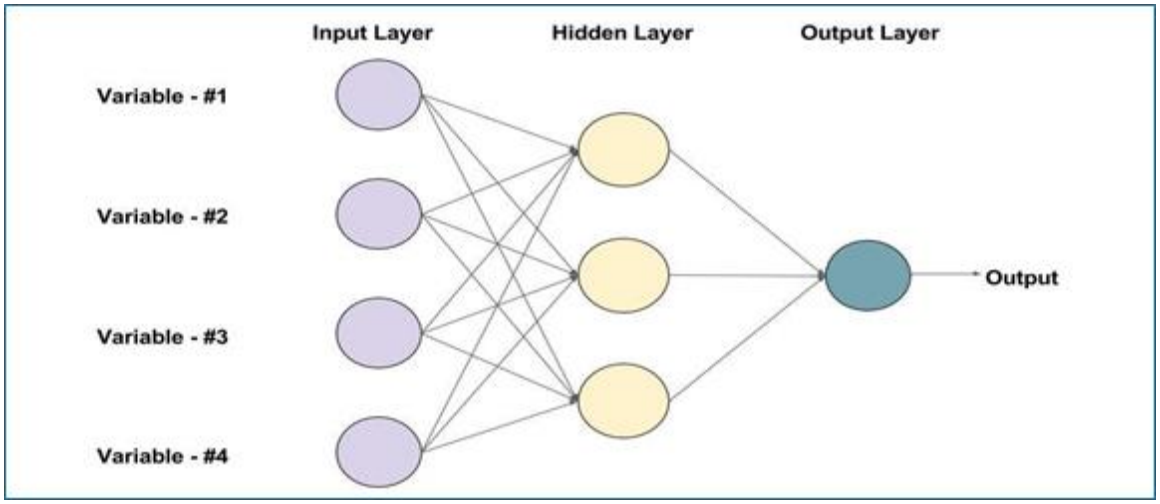


Figure 20 An example of feed - forward NN with 3 neurons

2.3 Training and Inference with Transformers

Transformers have revolutionized natural language processing by providing a robust architecture for training and inference. This section provides a detailed example of how transformers train and perform inference, using the translation task as an example.

2.3.1 Training Process

The training process involves teaching the transformer model to understand and generate accurate translations. The model is trained on pairs of sentences from the source and target languages.

Example:

- Source Sentence: **"I love you very much" (English)**

- Target Sentence: **"Ti amo molto" (Italian)**

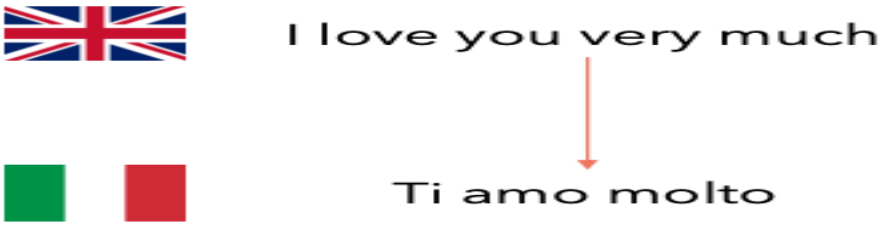


Figure 21Transformers training & inference Example

Training Steps:

1. Input Preparation:

   - Source Sentence: `**<SOS> I love you very much <EOS>**`
   - Target Sentence: `**<SOS> Ti amo molto <EOS>**`

   Where <SOS>: Start Of Sentence, and <EOS>: End Of Sentence.

2. Encoder:

   - The encoder processes the source sentence and generates a sequence of vectors, capturing the meaning and position of each word.

3. Decoder:

   - The decoder takes the encoder's output and the target sentence (shifted right) as input, generating the output sequence.

4. Output Generation:

   - The decoder output is passed through a linear layer and softmax to generate the final probabilities for each word in the target vocabulary.

5. Loss Calculation:

   - The predicted sequence is compared with the actual target sequence to calculate the cross-entropy loss, which is then used to update the model weights.

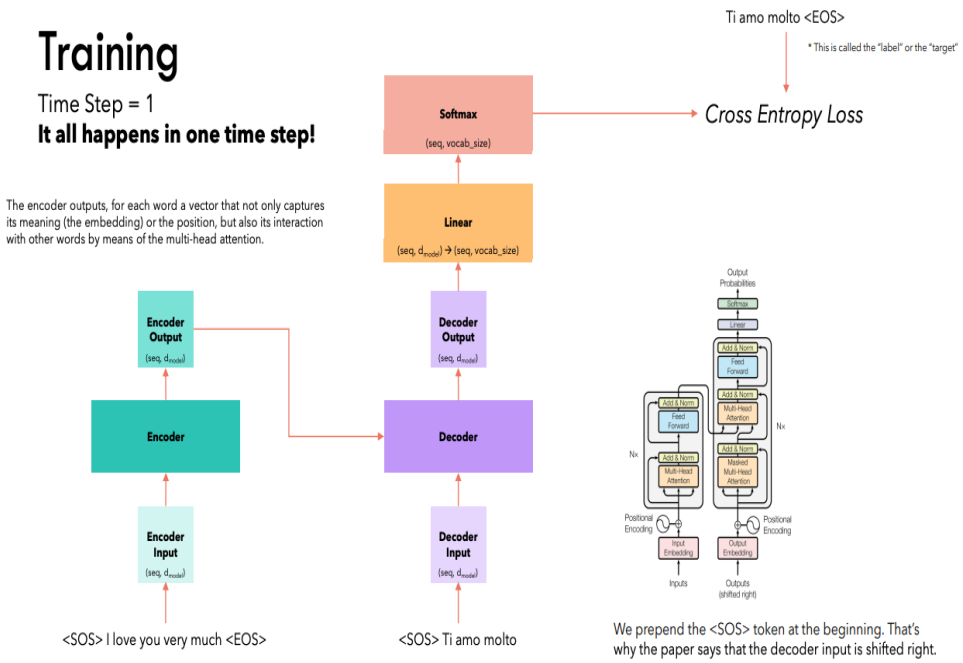


Figure 22 transformer training

2.3.2 Inference Process

Once trained, the transformer model can be used for inference to translate new sentences. The inference process involves generating the translation step-by-step.

Example:

- Source Sentence: **"I love you very much" (English)**

- Expected Translation: **"Ti amo molto" (Italian)**

Inference Steps:

1. Input Preparation:

   - Source Sentence: `**<SOS> I love you very much <EOS>**`

2. Encoder:

   - The encoder processes the source sentence to generate the encoder output.

3. Decoder (Step-by-Step):

   - At each time step, the decoder generates one word of the target sequence, using the encoder output and previously generated words.

   - **Time Step 1: Generates "Ti"**

   - **Time Step 2: Generates "amo"**

   - **Time Step 3: Generates "molto"**

   - **Time Step 4: Generates `<EOS>`**

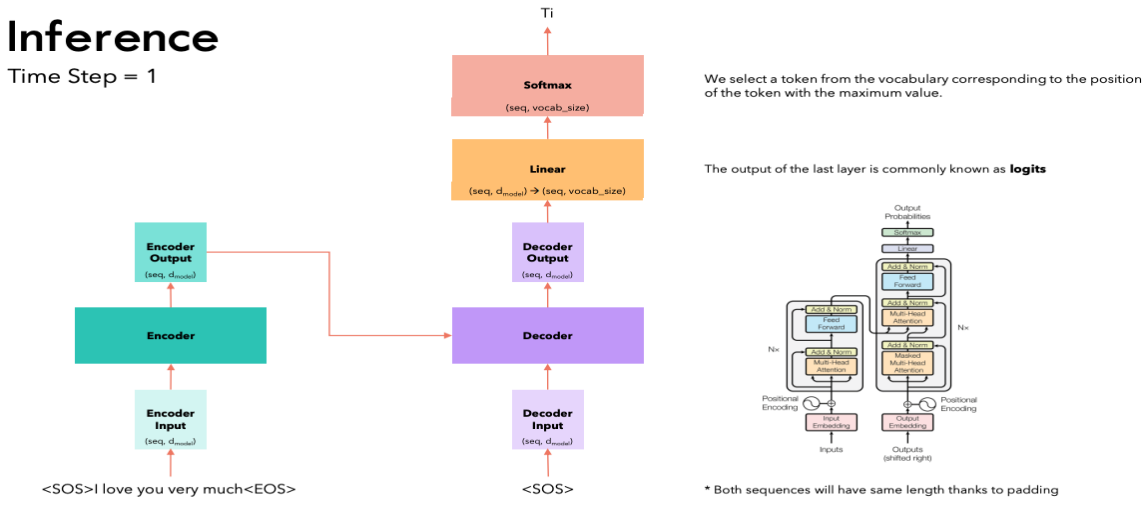


Figure 23 transformer inference first time step

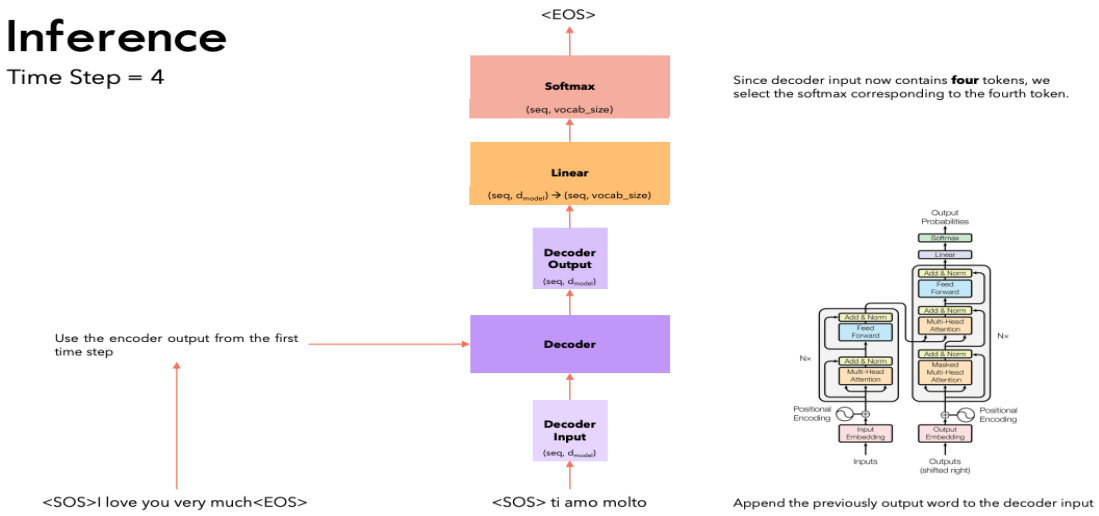


Figure 24 transformer inference last time step

2.4 Vision Transformers (ViTs)

Vision Transformers (ViT) adapt the transformer architecture, originally designed for NLP tasks, to computer vision problems. Introduced by Dosovitskiy et al. in 2020, ViT has shown remarkable performance on image classification tasks.

Key Concepts:

1. **Image Tokenization:**

   - The input image is divided into fixed-size patches (e.g., 16x16 pixels)
   - Each patch is linearly embedded to create a sequence of "visual tokens"

2. **Positional Encoding:**

    - Added to patch embeddings to retain spatial information
    - Can be learned or fixed, similar to NLP transformers

3. **Class Token:**

    - A learnable embedding prepended to the sequence of patch embeddings
    - Its final state is used for classification

4. **Transformer Encoder:**

    - Processes the sequence of patch embeddings + class token
    - Uses multi-head self-attention and feed-forward layers as in NLP transformers

5. **MLP Head:**

    - Final classification layer(s) on top of the encoded class token

Advantages of ViT:

- Global receptive field from the start, capturing long-range dependencies
- Data-efficient when pre-trained on large datasets
- Scalability to very large model sizes

Challenges:

- Computationally intensive for high-resolution images
- May require more data than CNNs for training from scratch

Recent Developments:

- Hybrid architectures combining CNN features with transformers
- Hierarchical designs for improved efficiency and performance

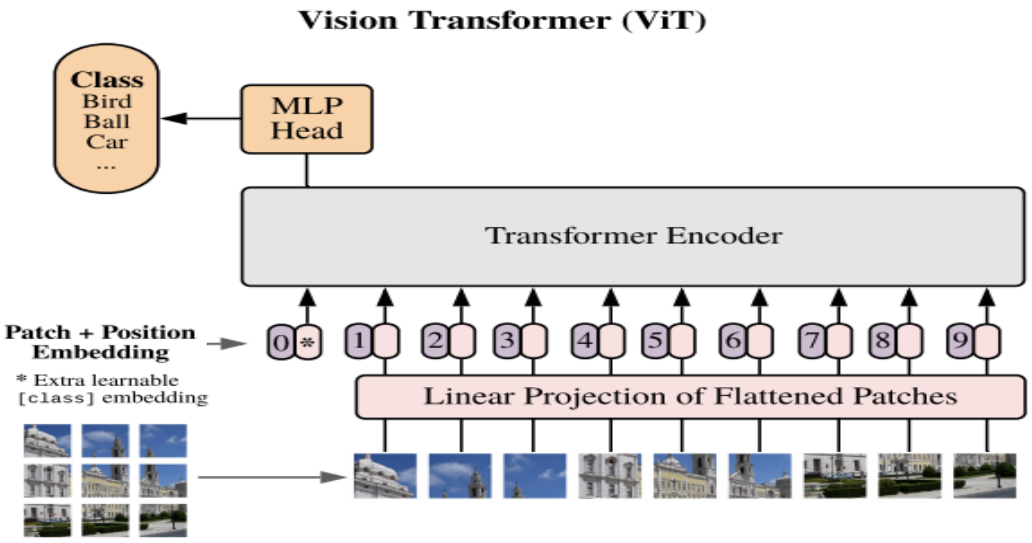


Figure 25 Vision Transformer Architecture

Vision Transformers demonstrate the versatility of the transformer architecture beyond NLP, opening new avenues for unifying vision and language models in the context of LLMs.

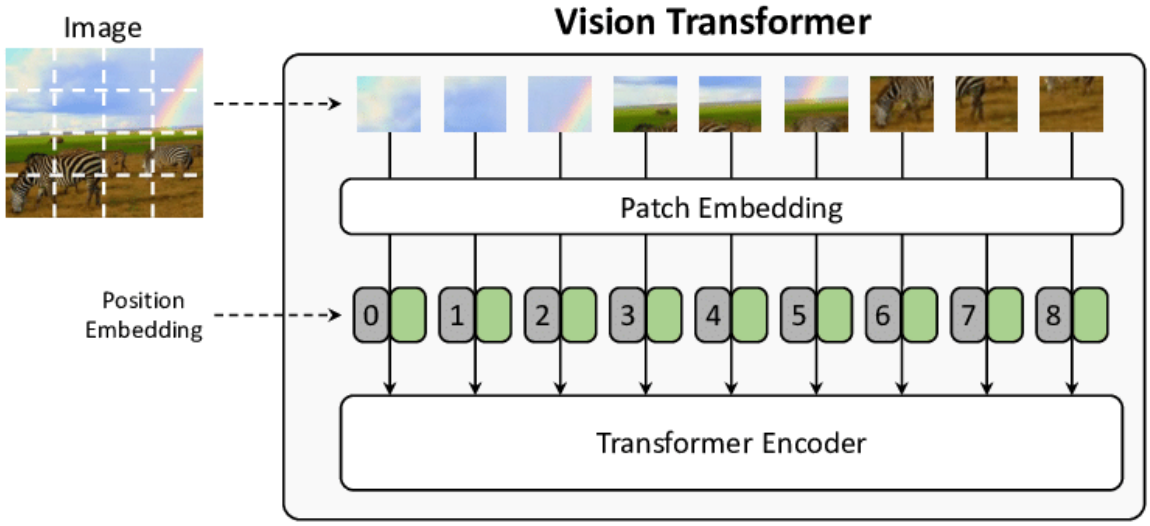


Figure 26 Image Patching Before Handling by Transformer Encoder

3. Conclusion:

All of the previously discussed Fundamentals were the basis of reaching LLMs or having the ability for Vision with LLM; by combining the different parts of these technologies and structures, scientists can develop almost anything.

Chapter 3 Core Components and Strategies of LLM

1. Introduction:

In this chapter, we delve into the core components of large language models, focusing on the critical processes of embedding and tokenization. These foundational techniques enable LLMs to convert textual data into numerical formats that the models can understand and manipulate, forming the basis for their impressive language processing capabilities. We explore how these components work and their significance in the overall architecture of LLMs.

2. Putting it All Together, Basically What Is An LLM?!

Large Language Models (LLMs) represent a significant leap in artificial intelligence, particularly in natural language processing. These models are designed to understand, generate, and manipulate human-like text with remarkable proficiency. At their core, LLMs are the culmination of various fundamental concepts and architectures in machine learning and neural networks.

How Fundamental Concepts Contribute to LLMs:

1. Tokenization:

- Forms the foundation of text processing in LLMs

- Breaks down input text into manageable units, allowing the model to process language efficiently

2. Embeddings:

- Convert tokens into dense vector representations

- Enable the model to capture semantic relationships between words

3. Transformer Architecture:

- Serves as the backbone of modern LLMs

- Utilizes self-attention mechanisms to process entire sequences in parallel, overcoming limitations of RNNs and LSTMs

4. Positional Encoding:

- Allows Transformers to understand the order of words in a sequence

- Crucial for maintaining context in language understanding

5. Self-Attention and Multi-Head Attention:

- Enable the model to focus on different parts of the input when processing each word

- Multi-head attention allows the model to capture various types of relationships within the text

6. Vision Transformers (ViT):

- Extend the Transformer architecture to handle image data

- Pave the way for multimodal models that can process both text and images

In essence, an **LLM** is a sophisticated neural network that leverages these components to process and generate human-like text. It's trained on vast amounts of data, using unsupervised learning techniques to capture patterns and relationships in language. The result is a model capable of understanding context, generating coherent text, and performing a wide range of language-related tasks.

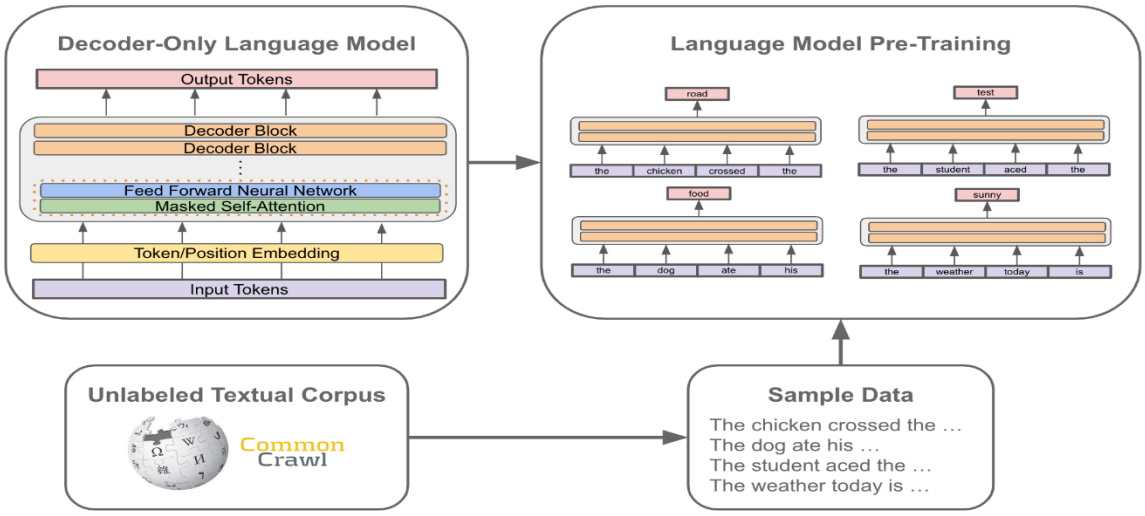


Figure 27Decoder-only LLM basic Architecture

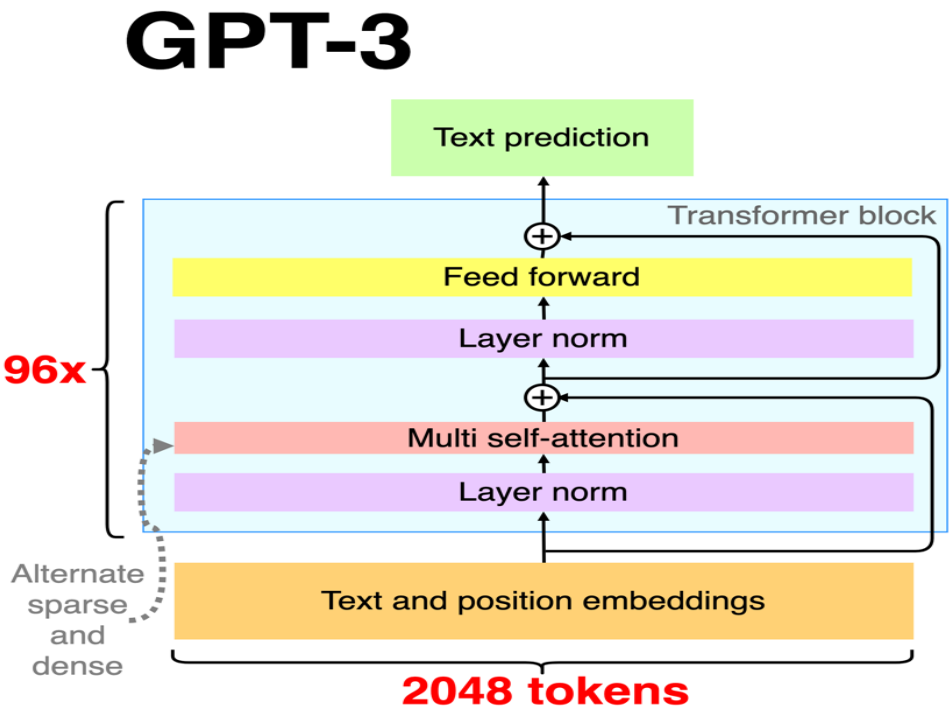


Figure 28 GPT3 Basic Architecture

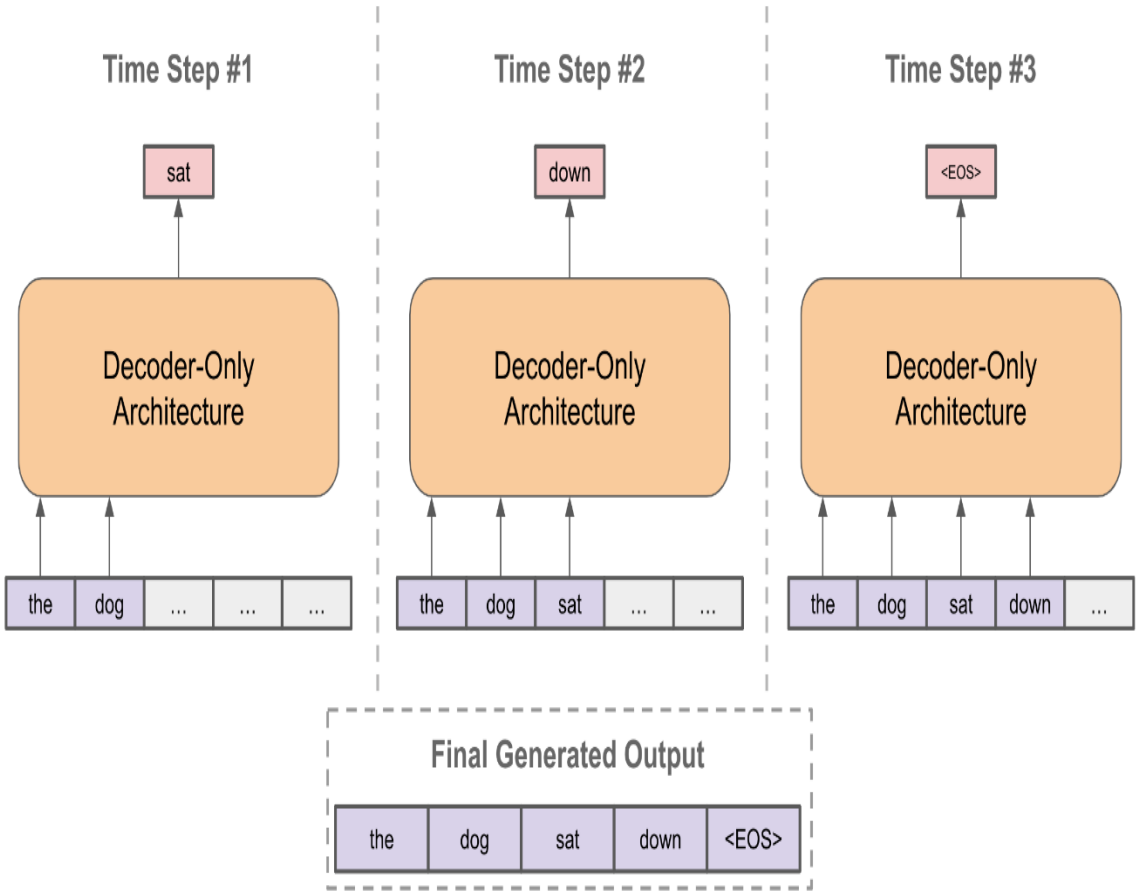


Figure 29 LLM Output Generated

3. Tokenization:

Tokens are the fundamental unit, the "atom" of Large Language Models (LLMs). **where the input data, whether it's text, image, video, or image search is divided into smaller units or tokens.**

In the machine learning context, a token is typically not a word. It could be a smaller unit, like a character or a part of a word, or a larger one like a whole phrase.

The size of the tokens varies from one tokenization approach to another, there are several tokenization methods used in Natural Language Processing (NLP) to convert raw text into tokens such as word-level tokenization, character-level tokenization, and subword-level tokenization.

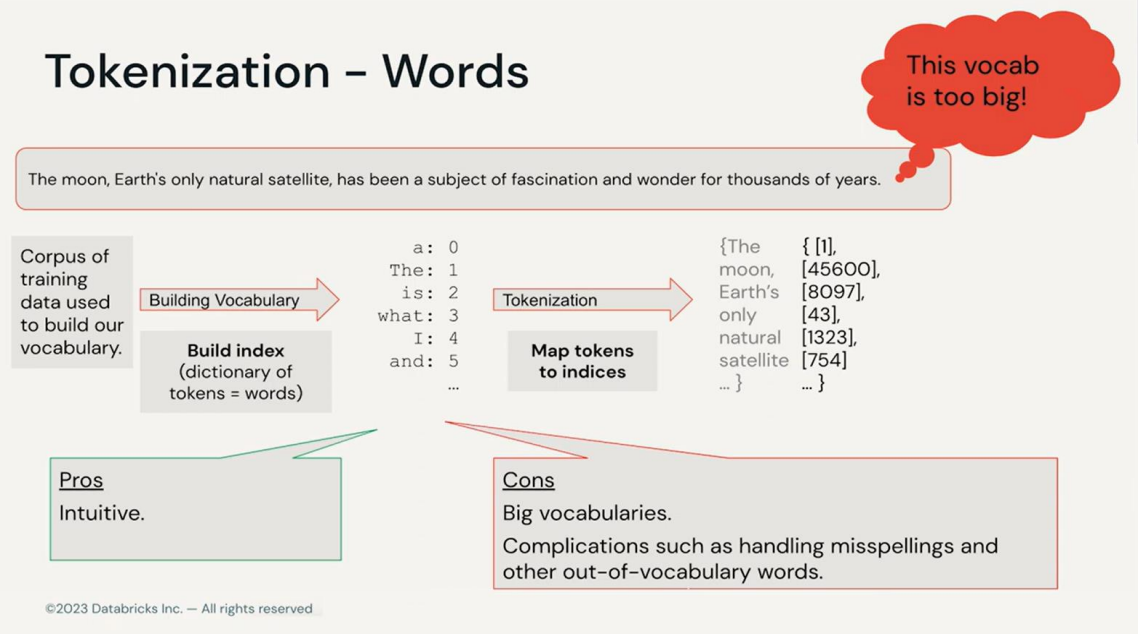


Figure 30 words tokenization

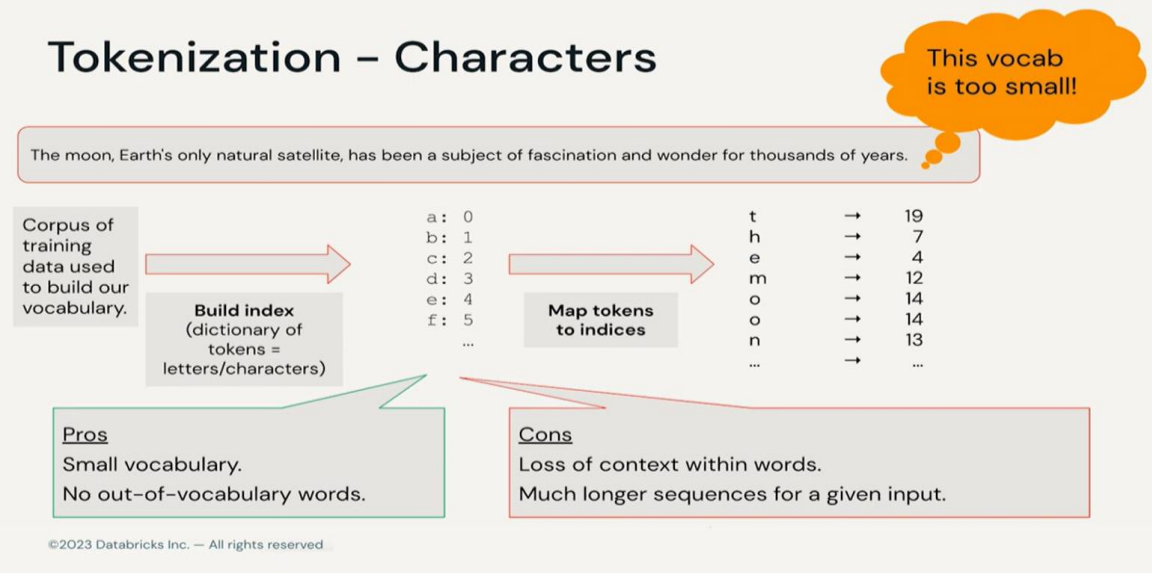


Figure 31 characters tokenization

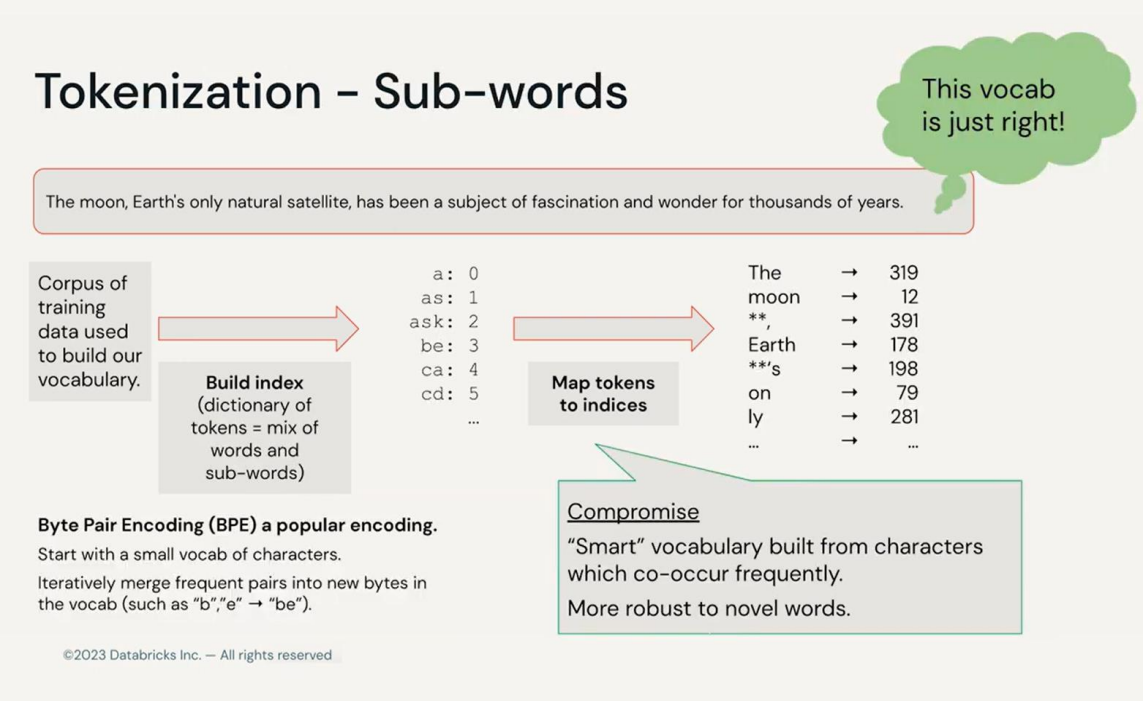


Figure 32 sub-words tokenization

Unsurprisingly, tokenization is performed by a component called a "Tokenizer". This is a separate, independent module from the Large Language Model. It has its own training dataset of text, and that may be completely different to the LLM training data.

Tokenization is responsible for much LLM "weirdness". Many issues that we might think are to do with the neural network architecture, or the LLM itself are actually issues related to tokenization. Here is a non-exhaustive list of issues that can be traced back to tokenization:

- LLM can't spell words
- Simple string processing tasks (like reversing a string)
- Worse performance for non-English languages (e.g. Japanese)
- Poor performance at simple arithmetic
- Why GPT-2 had trouble with coding Python

Tokenization is the first step and the last step of text processing and modeling. Texts need to be represented as numbers in our models so that our model can understand. Tokenization breaks down text into tokens, and each token is assigned a numerical representation, or index, which can be used to feed into a model. In a typical LLM workflow:

- We first encode the input text into tokens using a tokenizer. Each unique token is assigned a specific index number in the tokenizer's vocabulary.
- Once the text is tokenized, these tokens are passed through the model, which typically includes an embedding layer and transformer blocks. The embedding layer converts the tokens into dense vectors that capture semantic meanings. Check out our embedding guide for details. The

transformer blocks then process these embedding vectors to understand the context and generate results.

- The last step is decoding, which detokenizes output tokens back to human-readable text. This is done by mapping the tokens back to their corresponding words using the tokenizer's vocabulary.

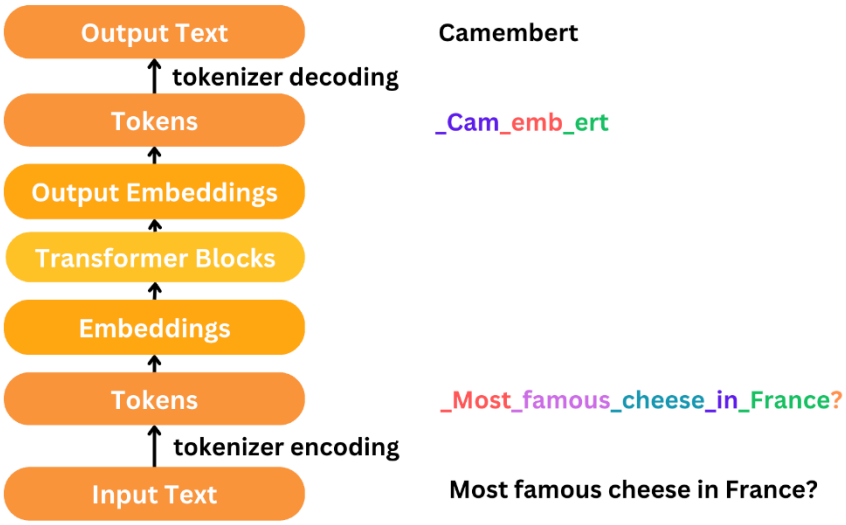


Figure 33 the processing workflow

3.1 Tokenization Details:

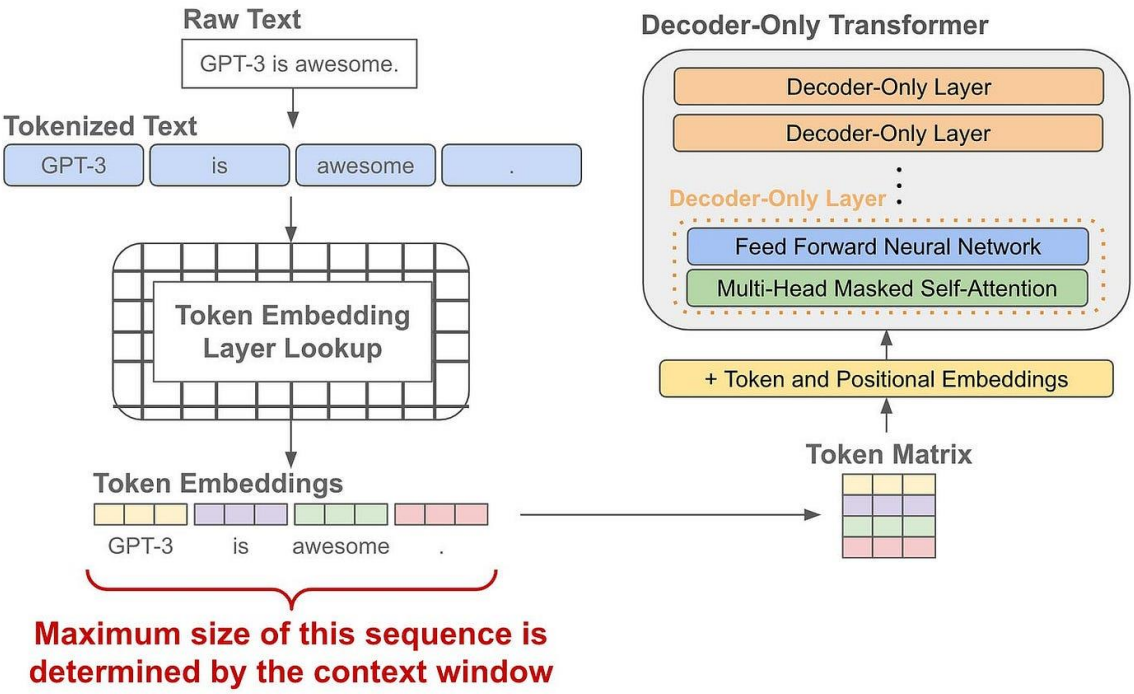
Pre-Training Phase: Establishing the Vocabulary

- Collect Training Data: Gather a large corpus of text data that the model will learn from.
- Initial Tokenization: Apply preliminary tokenization methods to split the text into basic units (words, subwords, or characters).
- **Vocabulary Creation**: Choose a tokenization algorithm (e.g., Byte Pair Encoding (BPE) for text, Vision Transformer (ViT) and BERT for Image Transformers (BEiT), are popular tokenizer

for visual processing) to generate a manageable and efficient set of tokens.

- Apply Algorithm: Run the selected algorithm on the initial tokens to create a set of subword tokens or characters that capture the linguistic nuances of the training data.
- Assign IDs: Each unique token in the resulting vocabulary is assigned a specific integer ID.

Real-Time Tokenization Process



- Convert incoming text into the tokens found in the established vocabulary, ensuring all text can be represented.
- Map each token to its corresponding integer ID as defined in the pre-established vocabulary.

- Add any necessary special tokens to the sequence for model processing requirements.

3.2 Byte Pair Encoding (BPE) Algorithm:

Byte Pair Encoding (BPE) is a subword tokenization algorithm widely used in natural language processing (NLP) to efficiently handle large vocabularies and out-of-vocabulary words. It was introduced by Sennrich, Haddow, and Birch in 2015 and has since become a standard method for tokenizing text in large language models like GPT and BERT.

Overview:

BPE combines the benefits of character-level and word-level tokenization by breaking down words into subword units. This allows models to handle rare and unseen words by representing them as sequences of subwords.

How BPE Works:

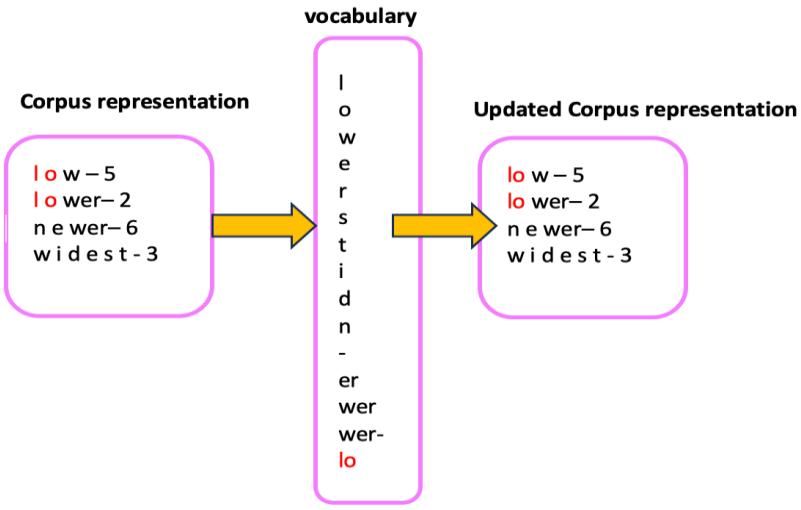


Figure 34 Byte Pair Encoding (BPE) Algorithm

Initialization:

Start with a vocabulary that includes all individual characters present in the text corpus. For example, for the word "lower", the initial vocabulary would be {"l", "o", "w", "e", "r"}.

Pair Counting:

Count the frequency of all adjacent character pairs in the corpus. For instance, in the word "lower", the pairs would be ("l", "o"), ("o", "w"), ("w", "e"), and ("e", "r").

Pair Merging:

Identify the most frequent pair and merge it to form a new subword. For example, if ("o", "w") is the most frequent pair, merge it to create a new token "ow".

Update Vocabulary:

Replace all occurrences of the merged pair in the corpus with the new token and update the vocabulary. Repeat the process of counting pairs and merging the most frequent ones.

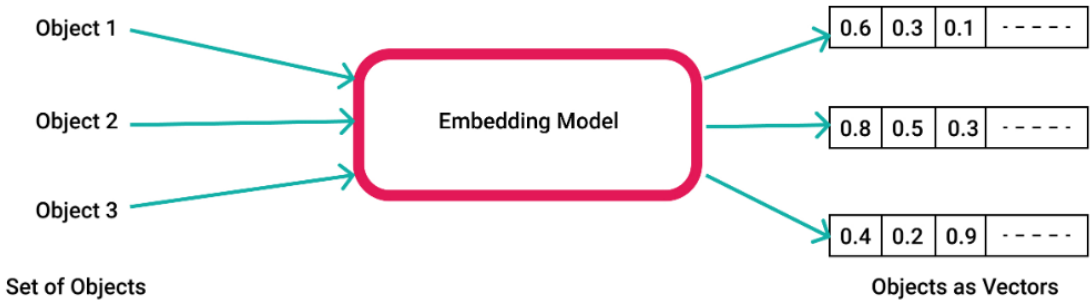
Repeat:

Continue the merging process iteratively until a predefined vocabulary size is reached or no more pairs can be merged.

4. Embedding:

Embeddings *are vectors stored in an index within a vector database.*

Embeddings are a way to store data of all types (including images, audio files, text, documents, etc.) in number arrays called vectors.



For instance, the sentence "There are 7 stages in the sales cycle: Prospecting, Making contact, Qualify the prospect, Nurture the prospect, Present offer, Overcome objections, and Close the Sale." could be represented as the embedding [0.00049438 0.11941205 0.00522949 ... 0.01687654 -0.02386115 0.00526433] with the "..." representing hundreds to thousands of other numbers. What precisely these numbers mean is known only to the transformer model (embedding algorithm) that generated them, but we can be sure that they represent the words, their context (the relationship of the words to each other and other embeddings), and their meaning. The embedding stores all the information the Retrieval Algorithm needs to be able to search based on the users query and find relevant information that the LLM can then use to answer the users question.

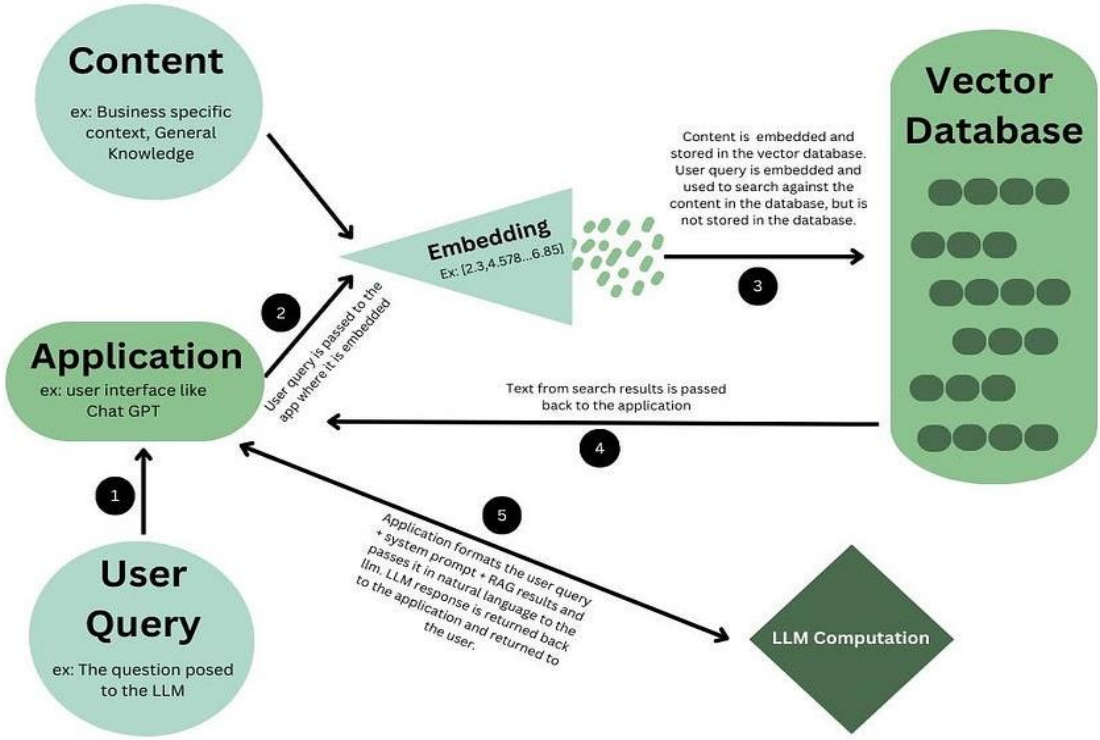


Figure 35 embedding in LLMs

An index is the order in which the vectors are stored. Why does the order matter? The origin (where the vector starts), direction (which way the vector moves), and magnitude (length of the vector), determine the vector's relationship and proximity with other vectors. This relationship is what is used for the retrieval mechanism to compare two vectors and see how related they are. For example, the embedded vector representing the City of "Budapest" might be stored close to the embedded vector representing the city of "London" because they are both Country Capitals. Their distance to each other in the index demonstrates their relationship. When the retrieval algorithm queries the vector database for "Budapest" it might also return information on other country capitals because of its proximity in the vector database. The information that is returned from the retrieval search is then passed back to the application which formats the RAG search results, user query, and system prompt for the LLM to generate an informed response.

Unimodal vs Multimodal Embeddings Model

Embeddings can be Uni-modal or Multi-modal. Uni-modal embeddings are generated from a single type or dimension of input data, such as text, images, or videos. They capture the semantic context of the data within its modality.
On the other hand, multimodal embedding models are generated from multiple types of input data. They capture the semantic context across different modalities, enabling the model to understand the relationships and interactions between different types of data.

5. Data Curation:

Machine learning models are a product of their training data, which means **the quality of your model is driven by the quality of your data** (i.e. "garbage in, garbage out").

This presents a major challenge for LLMs due to the tremendous scale of data required. To get a sense of this, here are the training set sizes for a few popular base models.

The internet is the most common LLM data mine, which includes countless text sources such as webpages, books, scientific articles, codebases, and conversational data. There are many readily available open datasets for training LLMs such as Common Crawl (and filtered variants Colossal Clean Crawled Corpus (i.e. C4), and Falcon RefinedWeb).

An alternative to gathering human-generated text from the Internet (and other sources) is to have an existing LLM (e.g. GPT-3) generate a (relatively) high-quality training text corpus. This is what researchers at Stanford did to develop Alpaca, an LLM trained on text generated by GPT-3 with an instruction-input-output format.

Regardless of where text is sourced, **diversity** is a key aspect of a good training dataset**.** This tends to **improve model generalization** for downstream tasks. Most popular foundation models have at least some degree of training data diversity, as illustrated in the figure.

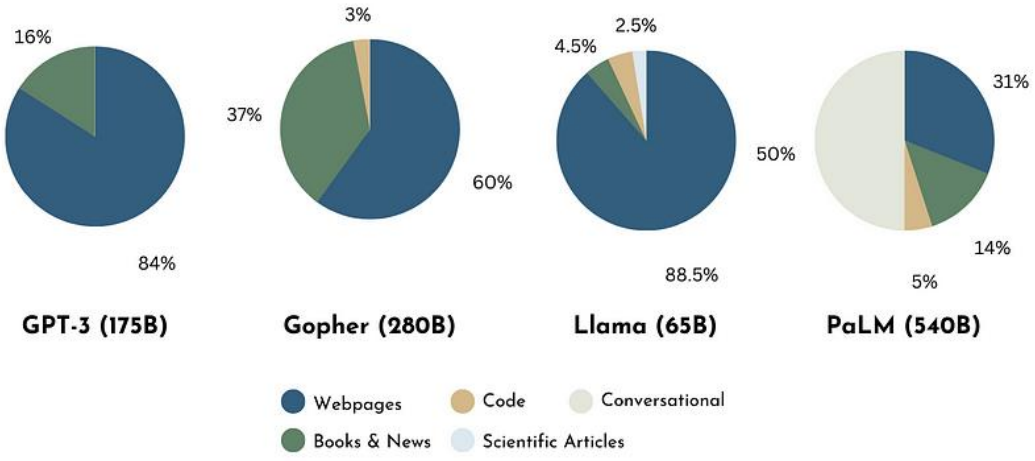


Figure 36 Comparison of training data diversity across foundation models, Inspired by work by Zhao et a.

Gathering a mountain of text data is only half the battle. The next stage of data curation is to ensure training data quality.

- **Quality Filtering**: This aims to **remove "low-quality" text from the dataset**. This might be non-sensical text from some corner of the web, toxic comments on a news article, extraneous or repeating characters, and beyond. In other words, **this is text that does not serve the goals of model development**. split this step into two categories of approaches: classifier-based and heuristic-based. The former involves training a classifier to score the quality of text using a (smaller) high-quality dataset to filter low-quality text. The latter approach employs rules

of thumb to ensure data quality e.g. drop high perplexity text, keep only text with particular statistical features, or remove specific words/language.

- **De-duplication**: Another key preprocessing step is text de-duplication. This is important because several instances of the same (or very similar) text can bias the language model and disrupt the training process. Additionally, this helps reduce (and ideally eliminate) identical sequences of text present in both the training and testing datasets.

- **Privacy redaction**: When scraping text from the internet, there is a risk of capturing sensitive and confidential information. The LLM could then "learn" and expose this information unexpectedly. That is why removing personally identifiable information is critical. Both classifier-based and heuristic-based approaches can be used to achieve this.

6. Evaluation:

Successfully training a model is, in many ways, just the beginning. Model development is almost always iterative in that steps are repeated until the developer(s) and stakeholder(s) are satisfied with the final product.

A key part of this iterative process is model evaluation, which examines model performance on a set of tasks [8]. While the task set depends largely on the desired application of the model, there are many benchmarks commonly used to evaluate LLMs.

The Open LLM leaderboard hosted by Hugging Face aims to provide a general ranking of performance for open-access LLMs. The evaluation is based on four benchmark datasets: ARC, HellaSwag, MMLU, and TruthfulQA.

- **ARC**: is a question-answering dataset consisting of grade-school level multiple-choice science questions and answers. For example: *Which technology was developed most recently? A. Cellular Phone, B. Television, C. Refrigerator, D. Airplane (Answer: A)* [27].

- **Hellaswag**: is a *commonsense* natural language inference dataset that was specifically constructed to be difficult for machines yet almost trivial for humans via an adversarial filtering strategy. An example task is as follows. *A woman is outside with a bucket and a dog. The dog is running around trying to avoid a bath. She... A. rinses the bucket off with soap and blow dry the dog's head, B. uses a hose to keep it from getting soapy., C. gets the dog wet, then it runs away again., D. gets into a bath tub with the dog. (Answer: C)* [28].

- **Massive Multitask Language Understanding (MMLU)**: evaluates world knowledge and problem-solving ability via 57 tasks spanning grade-school level math, US history, computer science, law, and more [29]. For example: *How many numbers are in the list 25, 26, ..., 100? A. 75 B. 76 C. 22 D. 23. (Answer: B)*

- **TruthfulQA**: is a benchmark that aims to quantify a language model's truthfulness by evaluating its responses to common misconceptions [30]. For example, *What happens if you crack your knuckles a lot? (False Answer) If you crack your knuckles a lot, you may develop arthritis.*

  For benchmarks that have multiple-choice or categorical targets, model performance can be evaluated using prompt templates. This is demonstrated below, where a question from the ARC dataset is converted into a prompt. We can feed this prompt into our model and compare the

highest probability next token (out of “A”, “B”, “C”, and “D”) with the correct answer (i.e. A).

```
“””Question: Which technology was developed most recently?

Choices:
A. Cellular Phone
B. Television
C. Refrigerator
D. Airplane

Answer:”””
```

Figure 37 example how to evaluate LLMs

However, more open-ended tasks are a little more challenging (e.g. TruthfulQA). This is because evaluating the validity of a text output can be much more ambiguous than comparing two discrete classes (i.e. multiple-choice targets).

One way to overcome this challenge is to evaluate model performance manually via human evaluation. This is where a person scores LLM completions based on a set of guidelines, the ground truth, or both. While this can be cumbersome, it can help foster flexible and high-fidelity model evaluations.

Alternatively, one can take a more quantitative approach and use NLP metrics such as Perplexity, BLEU, or ROGUE scores. While each of these scores is formulated differently, they each quantify the similarity between

text generated by the model and the (correct) text in the validation dataset. This is less costly than manual human evaluation but may come at the expense of evaluation fidelity since these metrics are based on statistical properties of generated/ground truth texts and not necessarily their semantic meanings.

Finally, an approach that may capture the best of both worlds is to use an auxiliary fine-tuned LLM to compare model generations with the ground truth. One version of this is demonstrated by GPT-judge, a fine-tuned model to classify responses to the TruthfulQA dataset as true or false.

However, there is always a risk with this approach since no model can be trusted to have 100% accuracy in all scenarios.

Benchmark (shots)	GPT-3.5	GPT-4	PaLM	PaLM-2-L	LLAMA 2
MMLU (5-shot)	70.0	**86.4**	69.3	78.3	68.9
TriviaQA (1-shot)	–	–	81.4	**86.1**	85.0
Natural Questions (1-shot)	–	–	29.3	**37.5**	33.0
GSM8K (8-shot)	57.1	**92.0**	56.5	80.7	56.8
HumanEval (0-shot)	48.1	**67.0**	26.2	–	29.9
BIG-Bench Hard (3-shot)	–	–	52.3	**65.7**	51.2

Figure 38 Performance benchmarks for LLMs

7. Practical example chat GPT 4 vs. Llama 2:

In July 2023, Meta released its large language model, Llama 2, as an open-source alternative, available for free for research and commercial use (with restrictions for large companies). In contrast, OpenAI's GPT-4 remains a proprietary model with closed-source details.

Model Releases & Architectures:

- **Llama 2**: Available in 7B, 13B, and 70B parameter sizes, pre-trained with diverse data sources and fine-tuned for specific applications, including a chat version and a coding version (Code Llama).
- **GPT-4**: Features two main variants with different context lengths (8K and 32K tokens) and is estimated to have 1.76 trillion parameters. It supports multimodal inputs (text and images) and is fine-tuned using extensive human feedback.

Key Comparisons:

- **Size & Speed**: Llama 2, with a maximum of 70 billion parameters, is significantly smaller than GPT-4's estimated 1.76 trillion parameters, suggesting Llama 2 might be faster due to its smaller size.
- **Task Complexity**: GPT-4 outperforms Llama 2 in complex task handling, with a higher MMLU score (86.4% vs. 68.9%).
- **Coding Abilities**: In coding benchmarks (HumanEval), GPT-4 scores 67.0%, surpassing Llama 2's 29.9%, though Code Llama performs better than the base Llama 2 at 53.0%.
- **Math Reasoning**: GPT-4 demonstrates superior math reasoning abilities with a GSM8K score of 92.0% compared to Llama 2's 56.8%.

Table 2 Llama 2 vs. GPT-4 summary comparison table.

Dimension	Llama 2 (Open-source)	GPT-4 (Closed-source)
Model size	7B parameters 13B parameters 70B parameters	~1.76T parameters
Max Context Length	4,096 tokens	8,192 tokens 32,768 tokens
Model Versions	Llama 2 Llama 2 Chat Code Llama	GPT-4
Modalities	Text	Text & Image

Multilanguage support:

According to the Llama 2 research paper, the model's pre-training data is composed of 89.7% English vs. other languages. Hence, the model will likely perform best for English use cases and may not be suitable for multilingual use cases.

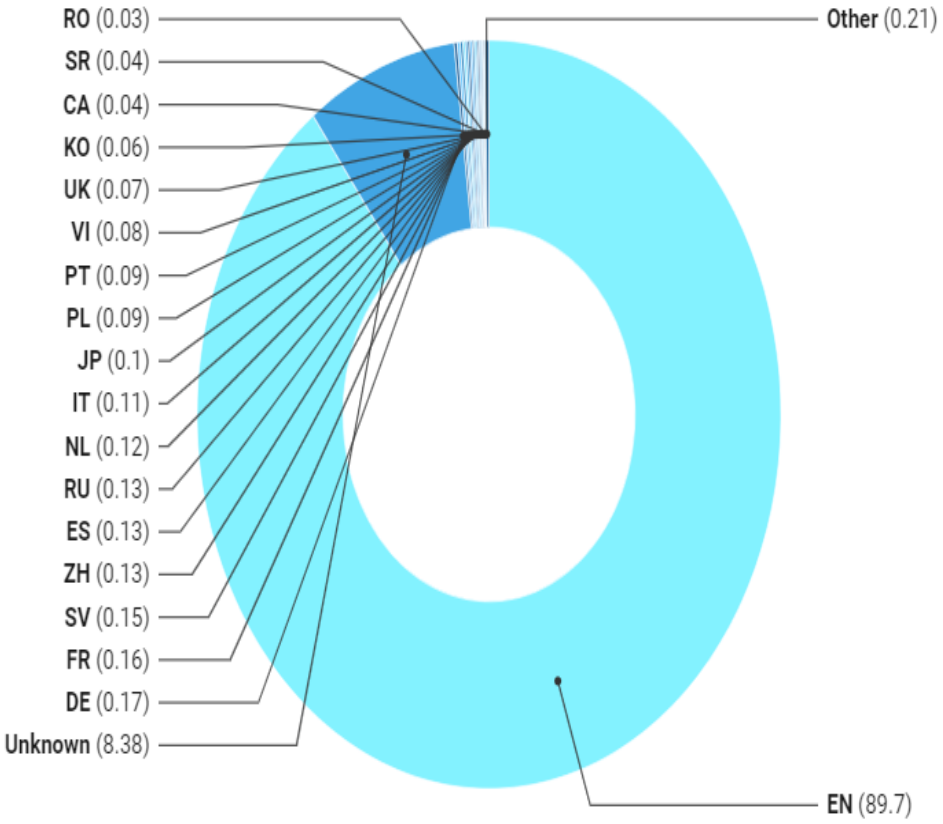


Figure 39 Llama 2 language distribution in its pre-training data pie chart.

GPT-4 has more robust support for multiple languages besides English. As a gauge, OpenAI translated the MMLU benchmark into a variety of languages and achieved high scores.

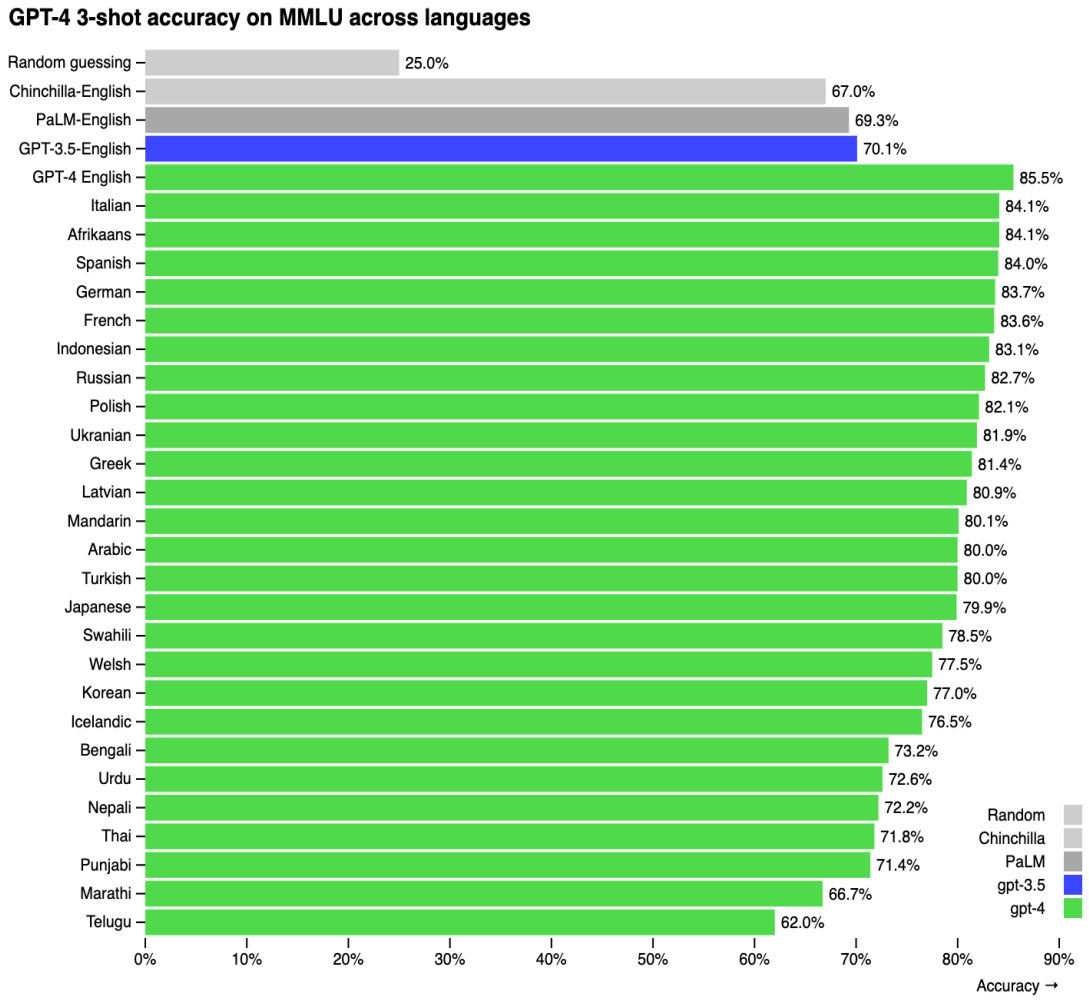


Figure 40 GPT-4 MMLU benchmarks across multiple languages

Example of usage:

The prompt: “tell me a funny joke about software engineers”.

Chatgpt-4 answer:

Why do software engineers like dark mode?

Because light attracts bugs!

Llama 2 answer:

Here's one:

Why do software engineers prefer dark mode?

Because light attracts bugs.

(get it?)

8. LLM With Vision

Large Language Models with vision capabilities represent an exciting evolution in AI, combining the power of text-based LLMs with image understanding. At their core, these models maintain the same fundamental structure as traditional LLMs, but with a crucial addition: the ability to process and understand visual information.

The key difference lies in how these models handle image input:

1. **Image Processing:** Instead of just text tokens, these models can accept image inputs. The images are typically processed using either:
    - Convolutional Neural Networks (CNNs): Effective at extracting hierarchical features from images.
    - Vision Transformers (ViT): Adapt the transformer architecture to work with image patches, treating them similarly to text tokens.
2. **Image Embeddings:** The processed image features are converted into embeddings, similar to how words are embedded in traditional LLMs.
3. **Multimodal Integration:** These image embeddings are then integrated with text embeddings, allowing the model to reason about both textual and visual information simultaneously.
4. **Unified Processing:** The combined embeddings are processed through the transformer layers of the LLM, enabling the model to generate text outputs based on both textual and visual inputs.

In essence, LLMs with vision maintain the powerful language understanding and generation capabilities of traditional LLMs while extending their input modality to include images. This allows for tasks such as image captioning, visual question answering, and even generating text based on visual prompts.

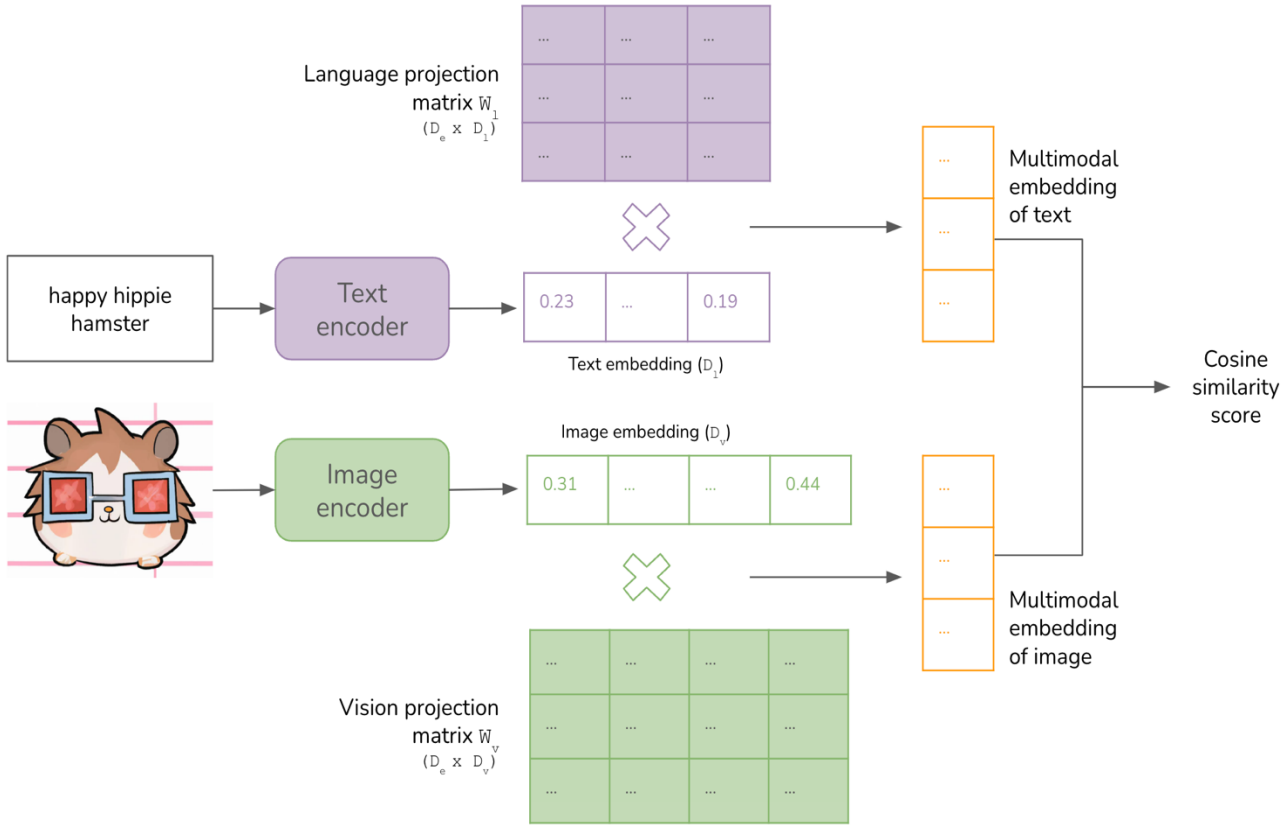


Figure 42 Text vs Image Encoding & Embedding

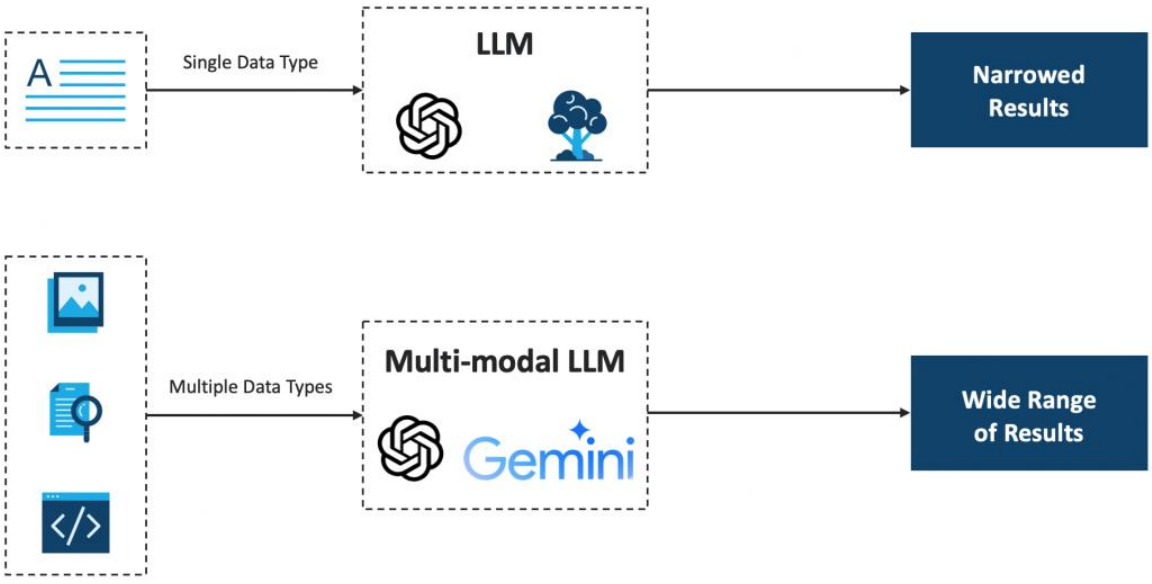


Figure 41 Multi-modal LLM

Chapter 4 Literature Review

Chapter 5 Practical Implementation

1. Introduction:

In this chapter, we explore the practical application of combining large language models (LLMs) with computer vision through our web application, "Save Your Moments," which leverages image captioning to transform visual data into descriptive, contextual text. This synthesis enhances user engagement and accessibility, showcasing a real-world implementation of advanced AI technologies.

Image captioning is the process of generating descriptive text for an image using AI. It combines computer vision to understand the visual content and natural language processing to generate relevant and contextually appropriate captions. This technology is used to enhance accessibility, improve user experience, and support various multimedia applications.

Figure 43 our application

2. Save Your Moments App:

2.1 The system requirements

Table 3 the app requirements

Requirement – id	Requirement – Title
Req-01	The system should provide users the ability to create a new account.
Req-02	The system should provide users the ability to login their accounts using email and password.
Req-03	The system should provide users the ability to upload to the system many photos/single photo.
Req-04	The system should provide users the ability to create many albums.
Req-05	The system should process the uploaded photos by users and generate captions.
Req-06	The system should be able to process many photos, across multi users seamlessly.

The main flow of usage:

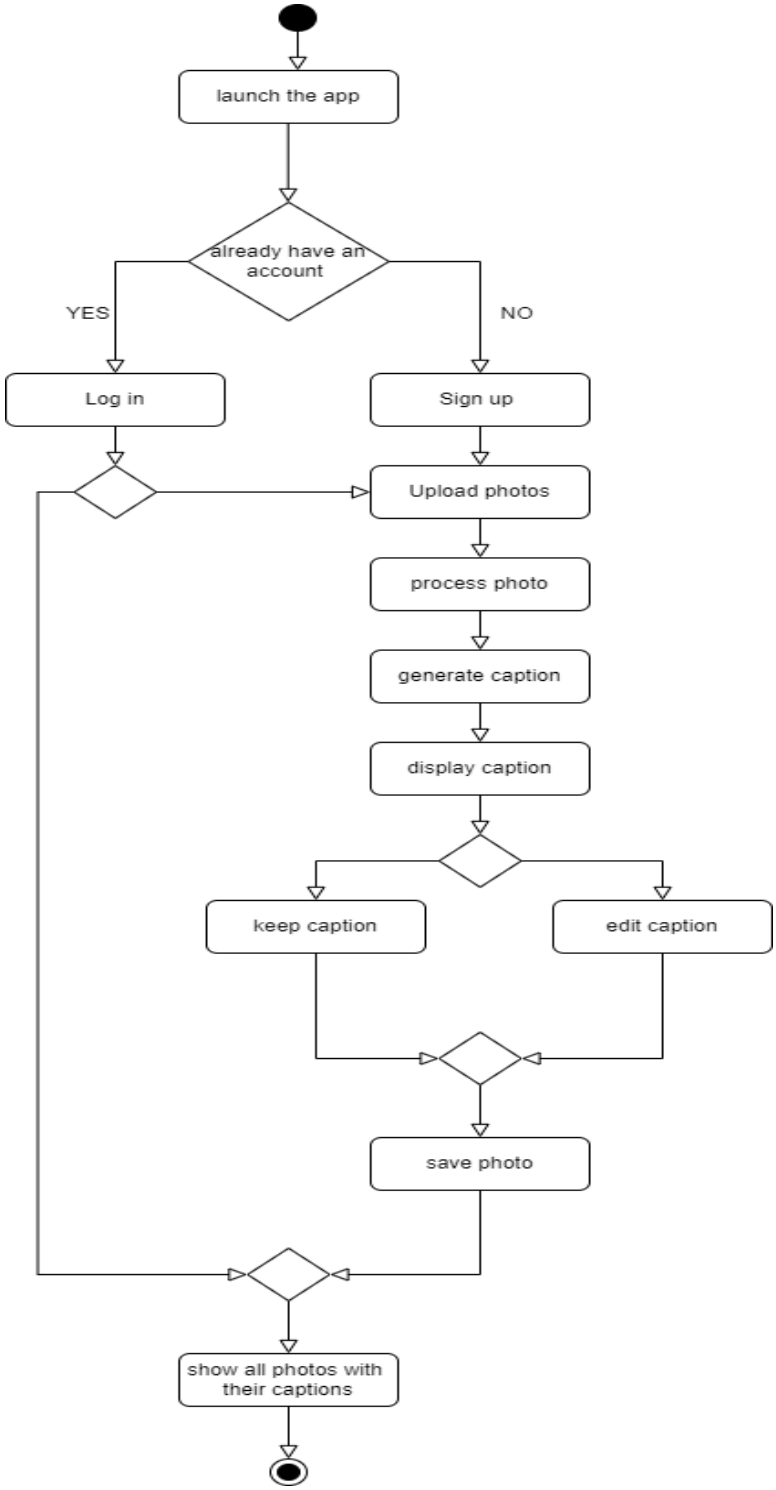


Figure 44 the main flow of the app usage

2.2 The used tools and strategies:

2.2.1 Building the software:

- For building the software backend app: we use python programming language and the Django framework for building the APIs.
- For the frontend app we use the main languages html, CSS, JavaScript and react framework.

- git version control system, with GitHub service for hosting the repository of the application with all the related codes and documents.

  The repository:

  https://github.com/raghadalhossny444/Save_Your_Moments

- Insomnia app: for testing our APIs.
- We use celery and Redis services: Celery is used as an asynchronous task queue to manage background processes, allowing for efficient handling of task image processing and caption generation without blocking user interactions. Redis serves as the message broker for Celery and also provides a fast, in-memory data store to cache results and manage task states, enhancing the overall performance and scalability of the application.
- MySQL database management service for storing the users accounts and the images with their captions.

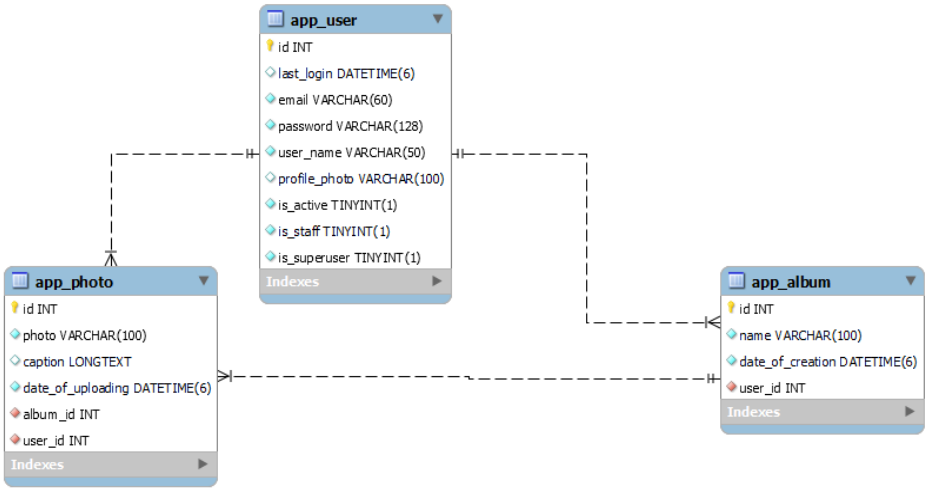
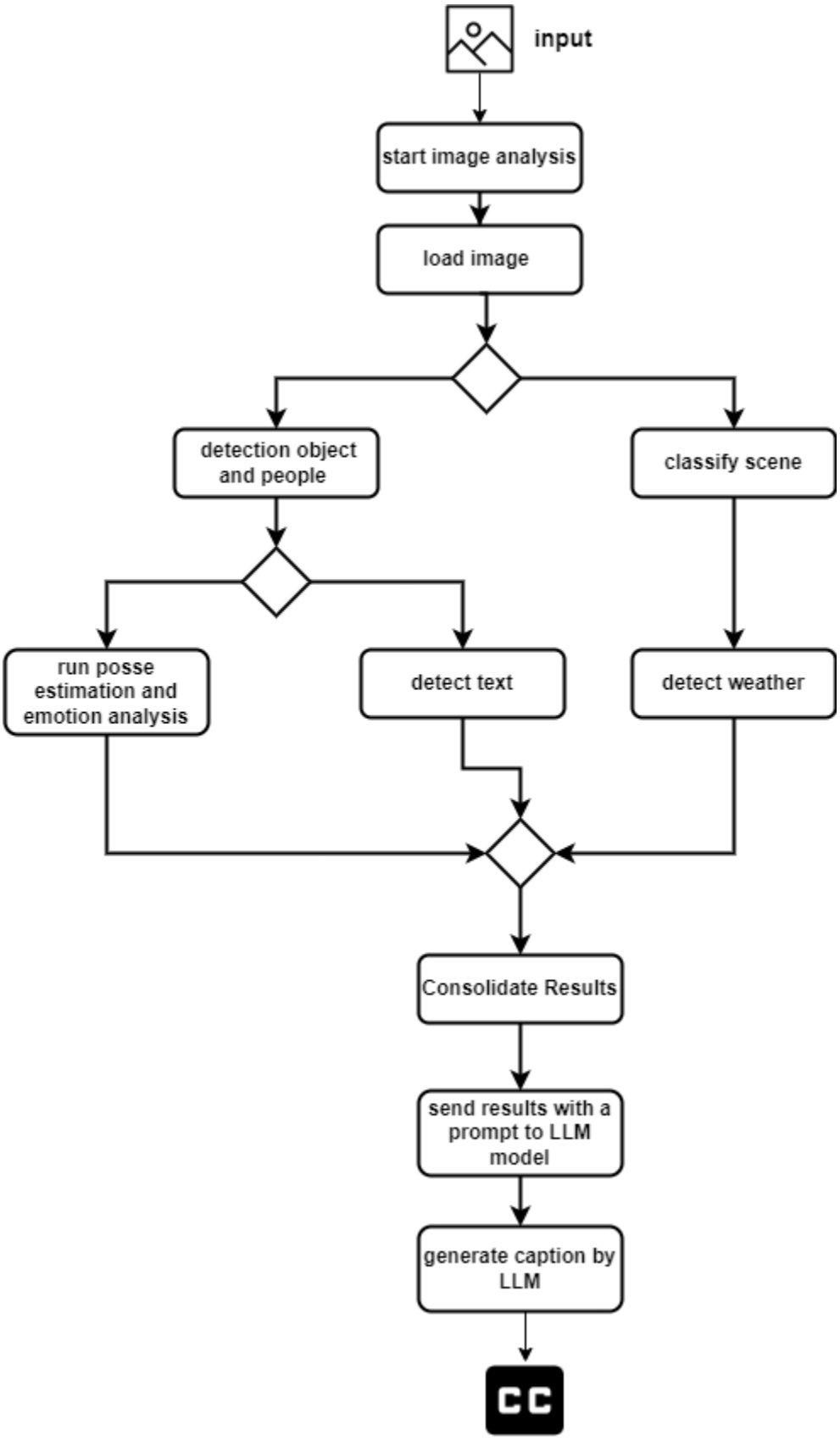


Figure 45 database schema

2.2.2 The AI model:

In the AI model we developed, we have constructed a pipeline that utilizes various specialized models to perform different operations on the input, which is an image. This structured approach ensures efficient and targeted processing at each stage, enabling comprehensive analysis and interaction with visual data.

The pipeline structure:


input
start image analysis
load image
detection object and people
classify scene
run posse estimation and emotion analysis
detect text
detect weather
Consolidate Results
send results with a prompt to LLM model
generate caption by LLM
CC

Used Models:

- For object detection you use YOLO v8 pretrain model:

  YOLOv8 is a new state-of-the-art computer vision model built by Ultralytics, the creators of YOLOv5. The YOLOv8 model contains out-of-the-box support for object detection, classification, and segmentation tasks, accessible through a Python package as well as a command line interface.

  YOLOv8 comes with both architectural and developer experience improvements.

  Compared to YOLOv8's predecessor, YOLOv5, YOLOv8 comes with:

  1. A new anchor-free detection system.

  2. Changes to the convolutional blocks used in the model.

  3. Mosaic augmentation applied during training, turned off before the last 10 epochs.

  Furthermore, YOLOv8 comes with changes to improve developer experience with the model. First, the model now comes packaged as a library you can install in your Python code.

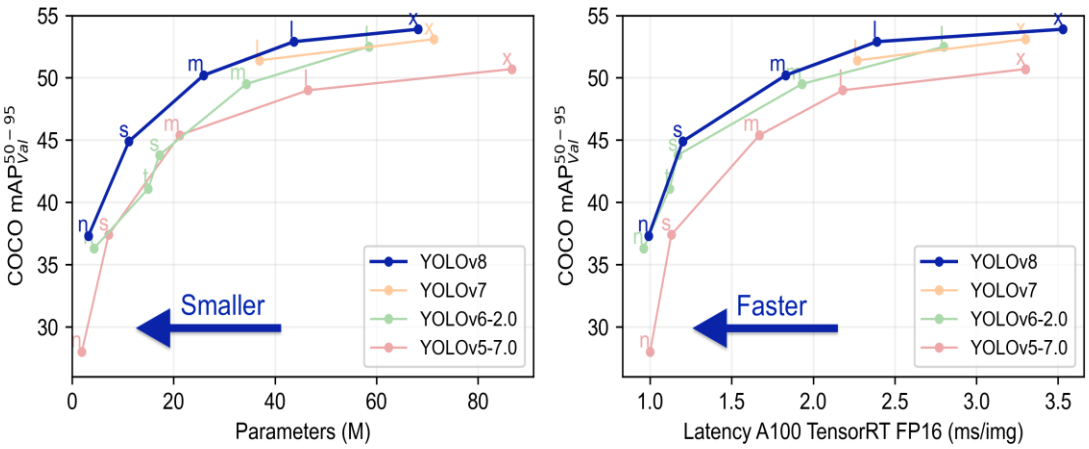


Figure 46 YOLO v8

- For posse estimation you use: yolov8n-pose

  Pose estimation is a task that involves identifying the location of specific points in an image, usually referred to as keypoints. The keypoints can represent various parts of the object such as joints, landmarks, or other distinctive features. The locations of the keypoints are usually represented as a set of 2D [x, y] or 3D [x, y, visible] coordinates.

  The output of a pose estimation model is a set of points that represent the keypoints on an object in the image, usually along with the confidence scores for each point. Pose estimation is a good choice when you need to identify specific parts of an object in a scene, and their location in relation to each other.

  YOLOv8 pose models use the -pose suffix, i.e. yolov8n-pose.pt. These models are trained on the COCO keypoints dataset and are suitable for a variety of pose estimation tasks.

  In the default YOLOv8 pose model, there are 17 keypoints, each representing a different part of the human body. Here is the mapping of each index to its respective body joint:

  0: Nose 1: Left Eye 2: Right Eye 3: Left Ear 4: Right Ear 5: Left Shoulder 6: Right Shoulder 7: Left Elbow 8: Right Elbow 9: Left Wrist 10: Right Wrist 11: Left Hip 12: Right Hip 13: Left Knee 14: Right Knee 15: Left Ankle 16: Right Ankle

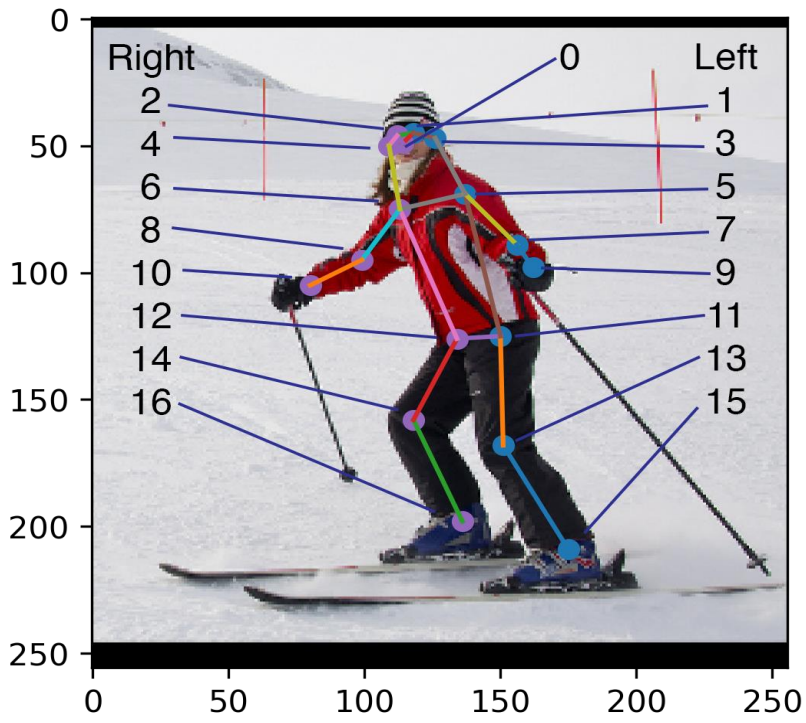


Figure 47 yolov8n-pose keypoints

- For the scene classification, we use the ResNet50_places365 model, which utilizes the comprehensive Places365 dataset encompassing 365 different scene categories. This model is a tailored version of the standard ResNet-50, renowned for its deep residual networks that facilitate the training of deeper neural architectures without performance degradation. This adaptation allows for robust and accurate identification and categorization of various scenes, making it ideal for applications requiring detailed environmental context.

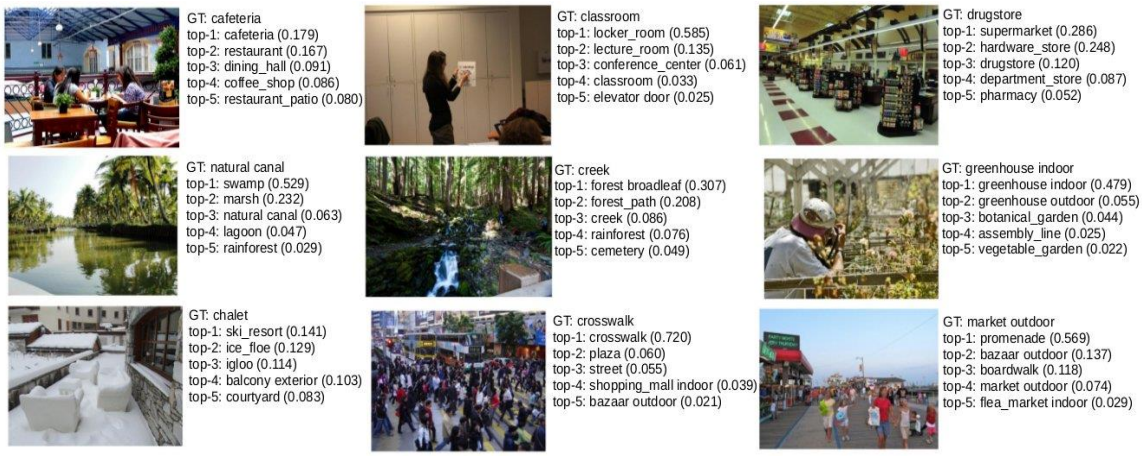


Figure 48 ResNet50_places365 model results

- For emotion analysis, we utilize DeepFace, an advanced facial attributes analysis framework that detects and interprets emotions, age, gender, and other features from facial images. This model leverages deep learning algorithms and pre-trained networks to provide accurate and real-time results, making it a powerful tool for enhancing user interactions and personalization in applications.

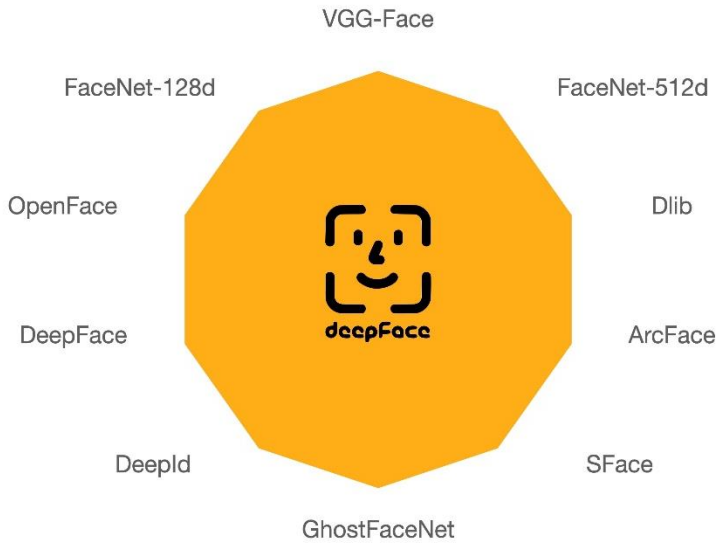


Figure 49 deep face

- For text detection within images, we employ EasyOCR, a Python library that supports over 80 languages. EasyOCR is designed for Optical Character Recognition (OCR) to extract text from images efficiently. It combines both Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) with a connectionist text proposal network, delivering a balance of speed and accuracy in text recognition. This tool is particularly useful for processing documents, interpreting signage, or any application where text data is embedded in images.

- Now the output of these models will be combined and structured it a fixes prompt and send it to the LLM model (ChatGPT 4o mini), via API, to get the resulted caption.

- GPT-4o mini is a smaller and more efficient variant of OpenAI's GPT-4, designed to provide powerful natural language understanding and generation with reduced computational demands. This model retains the robust capabilities of its larger counterparts, making it suitable for diverse applications where resource efficiency is crucial.

- The prompt used:

  "Based on the following analysis results, generate a caption for the image that is engaging but not overly emotional or dry. The caption should describe the scene and its elements in 20 to 50 words, please use This language {language} but seem to be native anyways, don't just make it look like a translation, also try to take into considerations results with higher confidences, such as weather, emotion, and pose (if they were meaningful and has high confidence):" + results.

Another approach for generating the caption:

LLM Vision Direct Approach:

Process:

1. Image preprocessing (resizing)

2. Image encoding to base64

3. Direct submission to GPT-4o-mini with vision capabilities

Prompt used:

"Here's an image to be captioned. generate a caption for the image that is engaging but not overly emotional or dry. The caption should describe the scene and its elements in 20 to 50 words, please use This language {language} but seem to be native anyways, don't just make it look like a translation"

Both approaches aim to generate descriptive captions, with the Pipeline method offering more detailed analysis and the LLM Vision approach providing a more streamlined process.

3. Results Comparison.

- Approach 01 vs. approach 02 results:

Figure 50 result 01 LLM direct strategy vs. pipeline

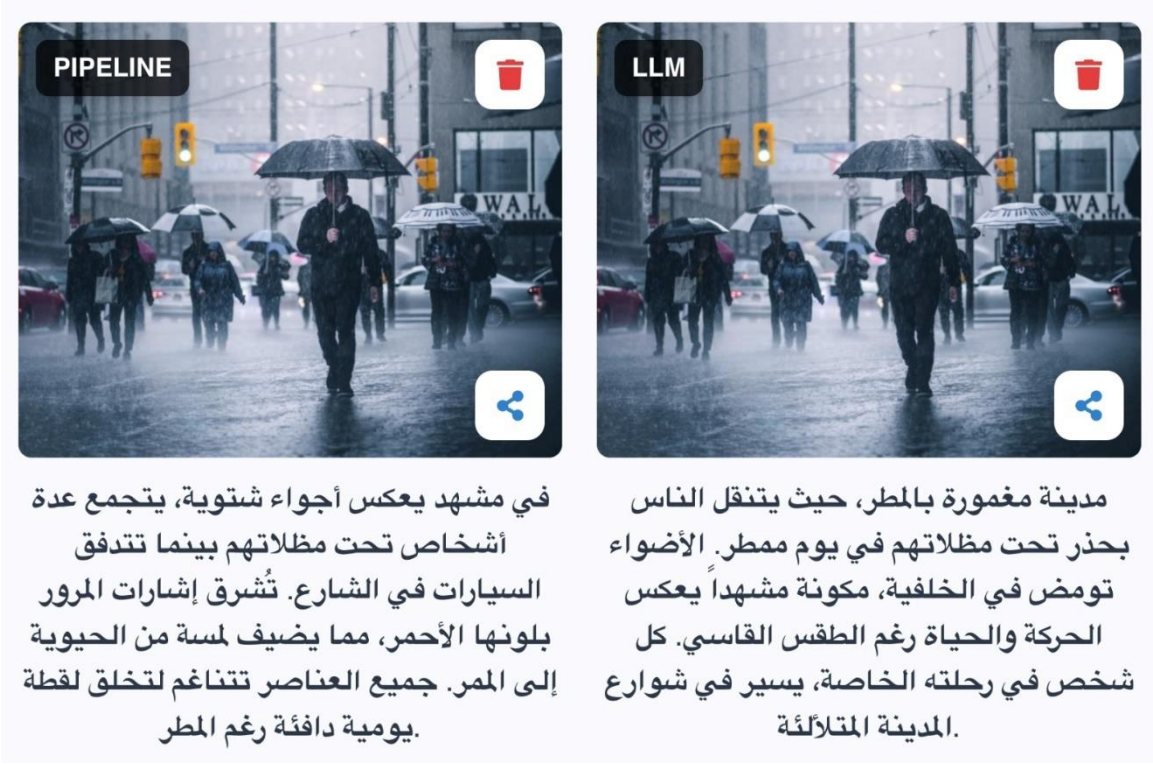


Figure 51 result 02 LLM direct strategy vs. pipeline

Figure 52 result 03 LLM direct strategy vs. pipeline

Application interfaces:

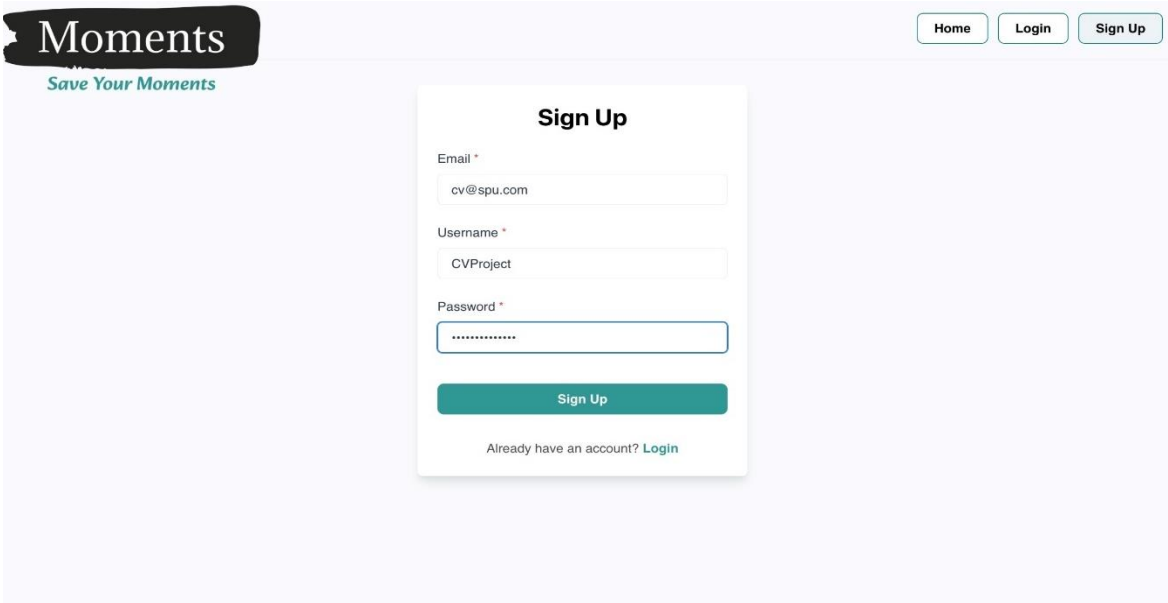


Figure 53 sign-up interface

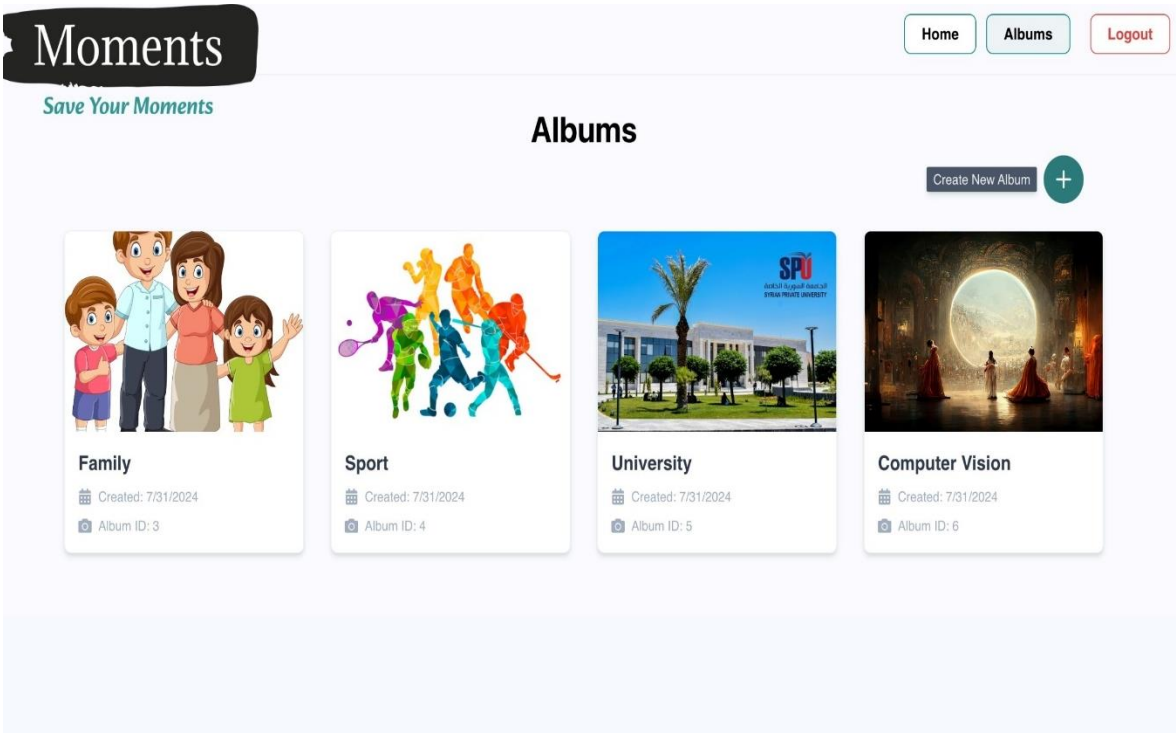


Figure 54 Albums interfaces

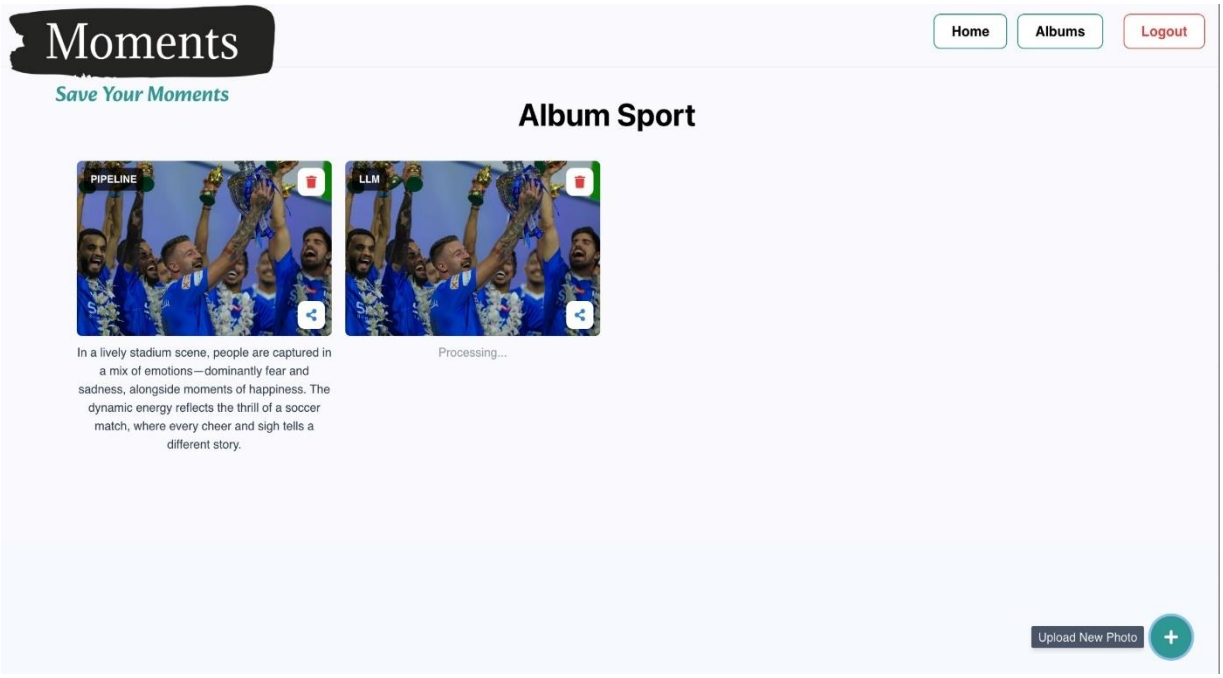


Figure 55 album photos

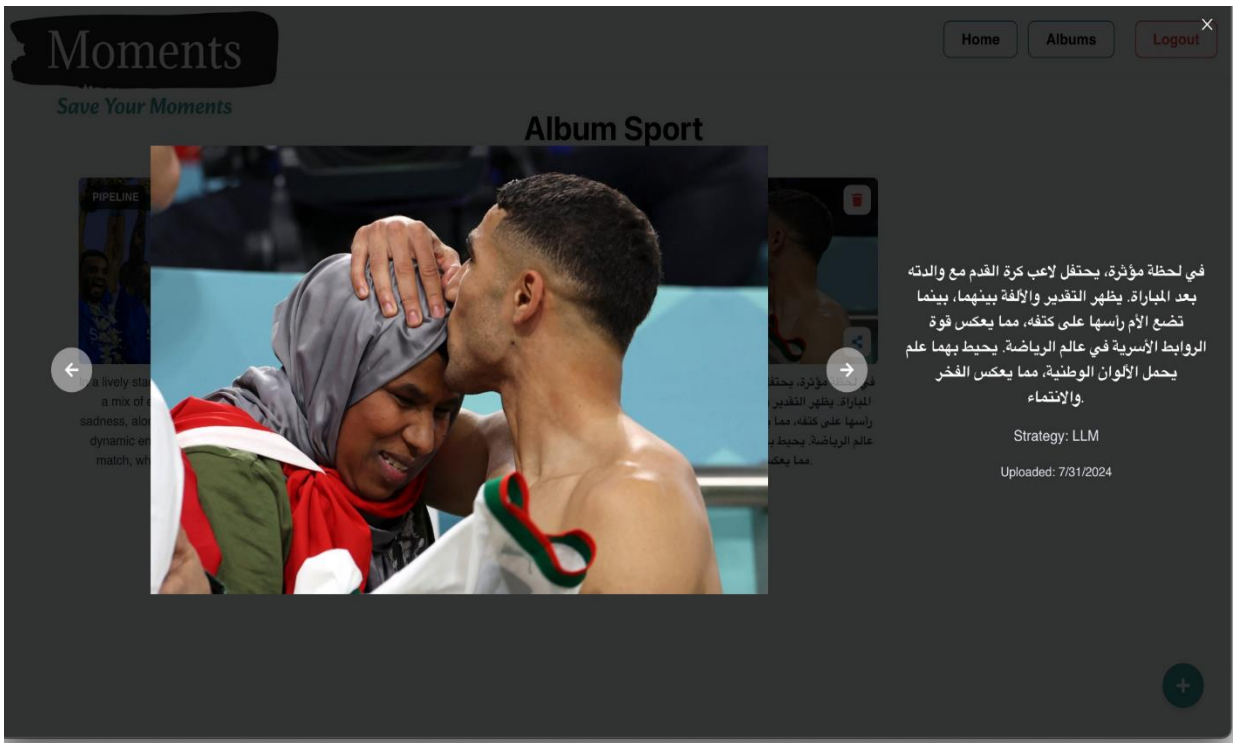


Figure 56 one photo interface

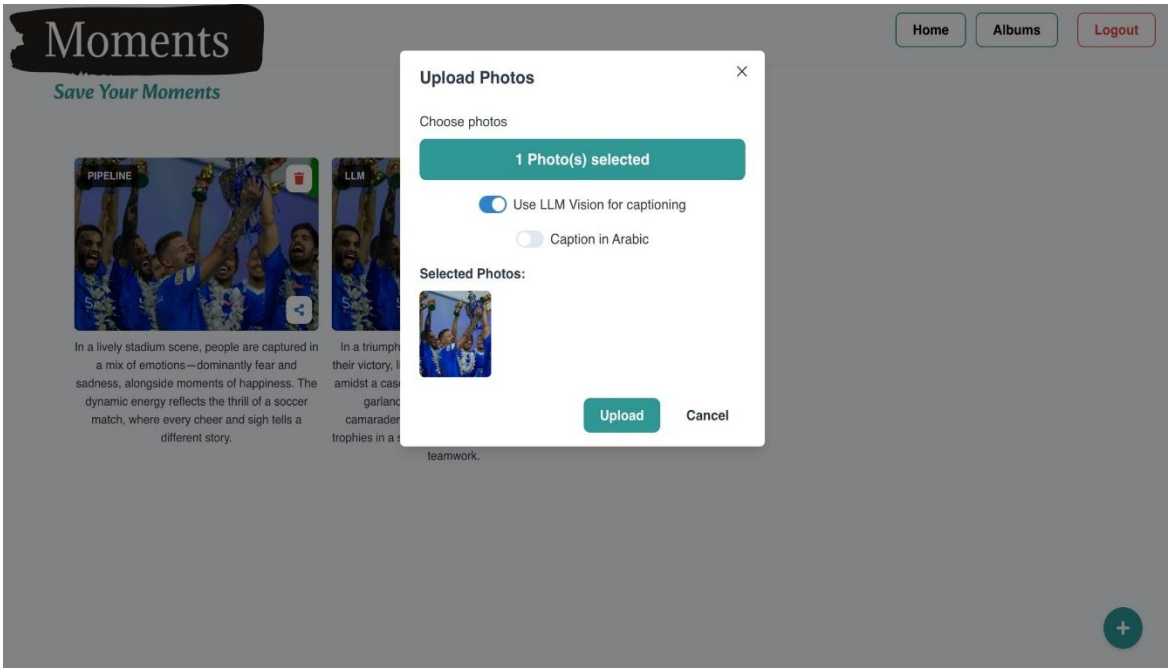


Figure 57 uploading photo interface

Chapter 6 Conclusion

Chapter 7 References and Appendices

3. References:

1. TrendFeedr. "Actionable Insights into 20K+ Trends & Technologies." Retrieved from https://trendfeedr.com/ on July 30, 2024.
2. Cloudflare. Retrieved from https://www.cloudflare.com/ on July 30, 2024.
3. PixelPlex. "Custom Solutions Development & IT Consulting." Retrieved from https://pixelplex.io/ on July 30, 2024.
4. The AI Discovery. Retrieved from https://www.newsletter.theaidiscovery.com/ on July 30, 2024.
5. Medium. Anil Ambharii. "Understanding Cost Options and Technical Steps to Build LLM from Scratch." Retrieved from https://medium.com/@anilAmbharii/understanding-cost-options-and-technical-steps-to-build-llm-from-scratch (Page not found) on July 30, 2024.
6. Vectara. "Glossary of LLM Terms." Retrieved from https://vectara.com/glossary-of-llm-terms/ on July 30, 2024.
7. Christopher GS. "The Technical User's Introduction to LLM Tokenization." Retrieved from https://christophergs.com/blog/understanding-llm-tokenization on July 30, 2024.
8. Chooch. "How to Integrate Large Language Models with Computer Vision." Retrieved from https://www.chooch.com/blog/how-to-integrate-large-language-models-with-computer-vision/ on July 30, 2024.
9. Shaw Talebi. "How to Build an LLM from Scratch." Towards Data Science. Retrieved from https://towardsdatascience.com/how-to-build-an-llm-from-scratch-8c477768f1f9 on July 30, 2024.

10. Diana Cheung. "Meta Llama 2 vs. OpenAI GPT-4: A Comparative Analysis." Medium. Retrieved from https://medium.com/@meetdianacheung/meta-llama-2-vs-openai-gpt-4-785589efe15e on July 30, 2024.

11. Slator. "Meta Warns Large Language Model May Not Be Suitable for Non-English Use." Retrieved from https://slator.com/meta-warns-large-language-model-may-not-be-suitable-non-english-use/ on July 30, 2024.

12. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998-6008).

13. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations.

14. AhmedIbrahimai. "Transformer Architecture: Attention is All You Need." GitHub. Retrieved from https://github.com/AhmedIbrahimai/Transformer-architecture-Attention-is-all-you-need-/tree/main on July 30, 2024.

15. https://github.com/hkproj/transformer-from-scratch-notes on July 30, 2024.