

UNIT 3: LARGE LANGUAGE MODELS (LLMs)

How transformer is suitable for NLP – Encoder only LLMs: BERT, Decoder only LLMs: GPT, Llama, MoE based Architecture, Significance of decoder only LLMs over encoder only LLMs - Natural Language Processing (NLP) - Translation, summarization, Named Entity Generation (NER) - Applications - Content Creation, Chatbot System.

I. HOW TRANSFORMER IS SUITABLE FOR NLP

Transformer models are a type of deep learning model that is used for natural language processing (NLP) tasks. They can learn long-range dependencies between words in a sentence, which makes them very powerful for tasks such as machine translation, text summarization, and question answering.

Transformers are well-suited for natural language processing (NLP) because they can:

- **Analyze long sentences**

Transformers can understand the context of a sentence, even when words at the beginning and end are related. For example, a transformer can understand that "I saw a statue standing in front of the store with binoculars" is ambiguous, and that the speaker was using binoculars to look at the statue.

- **Improve sentiment analysis**

Transformers can understand the context of a sentence to improve sentiment analysis. For example, "The food at the restaurant was good, but the service was terrible".

- **Process inputs in parallel**

Transformers are faster than RNNs and LSTMs because they process inputs in parallel.

- **Learn long-term dependencies**

Transformers use multi-head attention to learn long-term dependencies by looking at different representation subspaces at different positions.

Transformer models work by first encoding the input sentence into a sequence of vectors. This encoding is done using a self-attention mechanism, which allows the model to learn the relationships between the words in the sentence.

In 2017, Vaswani et al. published the paper "Attention is All You Need", which introduced a new transformer model that was much more efficient than the fast-weight controller. This new model, which is now simply called the "transformer", quickly became state-of-the-art for a wide range of natural language efficient (NLP) tasks, including machine translation, text summarization, and question answering.

The transformer has been so successful because it can learn long-range dependencies between words in a sentence. This is essential for many NLP tasks, as it allows the model to understand the context of a word in a sentence. The transformer does this using a self-attention mechanism,

which allows the model to focus on the most relevant words in a sentence when decoding the output tokens.

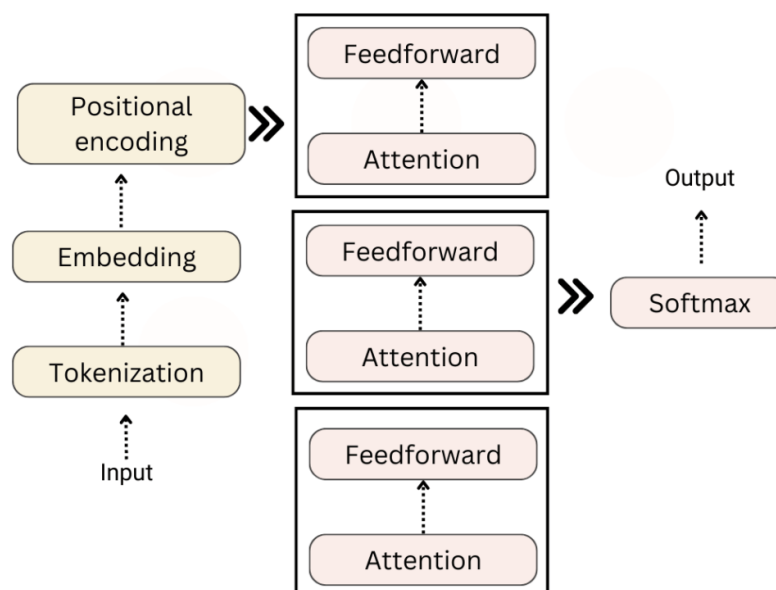
The transformer has had a major impact on the field of NLP. It is now the go-to approach for many NLP tasks, and it is constantly being improved. In the future, transformers are likely to be used to solve a wider range of NLP tasks, and they will become even more efficient and powerful.

Here are some of the key events in the history of transformers in neural networks:

- 1990: Jürgen Schmidhuber proposes the first transformer model, the “fast weight controller”.
- 2017: Vaswani et al. publish the paper “Attention is All You Need”, which introduces the transformer model.
- 2018: Transformer models achieve state-of-the-art results on a wide range of NLP tasks, including machine translation, text summarization, and question answering.
- 2019: Transformers are used to create large language models (LLMs) such as BERT and GPT-2.
- 2020: LLMs are used to create even more powerful models such as GPT-3.

The history of transformers in neural networks is still being written. It is an exciting time to be in the field of NLP, as transformers are making it possible to solve previously intractable problems.

How transformer models work



The encoder consists of a stack of self-attention layers. Each self-attention layer takes a sequence of vectors as input and produces a new sequence of vectors. The self-attention layer works by first computing a score for each pair of words in the input sequence. The score for a

pair of words is a measure of how related the two words are. The self-attention layer then uses these scores to compute a weighted sum of the input vectors. The weighted sum is the output of the self-attention layer.

The decoder consists of a stack of self-attention layers and a recurrent neural network (RNN). The self-attention layers work the same way as in the encoder. The RNN takes the output of the self-attention layers as input and produces a sequence of output tokens. The output tokens are the words in the output sentence.

The attention mechanism is what allows the transformer model to learn long-range dependencies between words in a sentence. The attention mechanism works by focusing on the most relevant words in the input sentence when decoding the output tokens.

For example, let's say we want to translate the sentence "I love you" from English to Spanish. The transformer model would first encode the sentence into a sequence of vectors. Then, the model would decode the vectors into a sequence of Spanish words. The attention mechanism would allow the model to focus on the words "I" and "you" in the English sentence when decoding the Spanish words "te amo".

Transformer models are a powerful tool for NLP, and they are constantly being improved. They are now the go-to approach for many NLP tasks, and they are constantly being improved.

Examples of NLP tasks using transformers

Here are a few examples to better understand the tasks transformers are used for and how they can help during model training and pre-training.

1. Using transformers can improve meaning clarity

Words at the beginning of a sentence may impact words at the end of a sentence. Transformers are particularly useful for analyzing longer sentences and connecting long-range dependencies.

Example: "I saw a statue standing in front of the store with binoculars."

The sentence itself is ambiguous. Was the statue looking at the speaker with binoculars or was the speaker looking at the statue? Transformer models look at the broader context to improve clarity.

Revised: "I saw a statue standing in front of the store with binoculars and zoomed in on it."

This example shows how transformers can capture the dependency and context of an input sequence, understanding that the speaker was using a pair of binoculars to look at the statue.

2. Improve sentiment analysis with transformers

Transformers can also help improve sentiment analysis by understanding the correct context.

Example: "The food at the restaurant was good, but the service was terrible."

This sentence uses mixed sentiments to describe different aspects of the restaurant experience. Without the proper context, some language models may struggle to assign the correct sentiment. With transformers, however, the context can be captured more effectively.

Revised: “The food at the restaurant was OK, but the service was terrible; I would not recommend it. Here, the overall sentiment is negative.

3. Fine-tuning responses during pre-training

Technologists can train transformer models using large data sets and then update them with smaller data sets for specific tasks to provide better answers. Teams at Capital One use this technique, known as fine-tuning, to create new experiences and [products such as a Slack bot](#).

Example: “What should I cook after work?”

This question could provide a variety of responses, but it may not match the question’s intent. By providing data on potential answers, you can narrow the focus to find the most likely response. The additional questions that might naturally flow from the first question help train models for better results.

Related question sets: “What are some one-pot meals I can cook?” or “What are some easy recipes to make?”

While these are similar, the subtle differences help refine the model during pre-training and improve future responses.

4. Translation

Transformers are especially valuable in translations, where models are trained to translate English into other languages.

For example, in the sentence, “I enjoy reading books in my free time,” the word enjoy has a nuanced meaning. A language model might struggle with the translation for enjoy, choosing “I like” (gusto) or “I love” (amo), which have very different meanings in Spanish.

A transformer model is more likely to produce a correct output that shows the accurate meaning: “Disfruto leyendo libros en mi tiempo libre.”

II. OVERVIEW OF LANGUAGE MODEL ARCHITECTURES

Let’s first familiarize ourselves with some architectural terms.

Encoder and Decoder

- Encoder: Processes and transforms input data into a condensed representation, capturing the essential information. In a translation task, an encoder takes an English sentence and converts it into a vector that represents its linguistic features and meaning.

- Decoder: Takes the encoded representation and generates an output, often in a different form. In the same translation task, the decoder takes the encoded representation of a sentence in English and generates its equivalent in French.

Encoder-Only Models

- Example: BERT-based models
- Pretraining Approach: Masked Language Modelling (MLM)
- Use Case: Tasks that require a deep understanding of input data. These models are effective for classification, sentiment analysis, and information extraction.

Decoder-Only Models

- Example: GPT, XLNet
- Pretraining Approach: Next Token Prediction
- Use Case: Generative tasks. They work by predicting subsequent text based on the provided context in an auto-regressive fashion. Their primary function is output generation without a separate encoding phase.

Encoder-Decoder Models

- Example: T5, BART, Google Gemini (Probably)
- Pretraining: Task-dependent
- Use Case: Tasks that involve both understanding and generating data. They first encode an input sequence into an internal representation and then decode this representation into an output sequence.

Comparing the purpose of these architectures, we can first easily exclude encoder-only models: They are typically pre-trained with MLM and do not necessarily help with generating output.

The decoder-only ones, on the other hand, make perfect sense: They are used to generate outputs and are pre-trained on Next Token Prediction tasks, which is exactly the task for most LLMs.

ENCODER ONLY ARCHITECTURE: BERT

BERT is an example of Encoder only transformer Architecture.

If you are working with a task which requires you to only analyze input data and reach a conclusion based on the analysis, a more appropriate architecture would be Encoder Only Architecture.

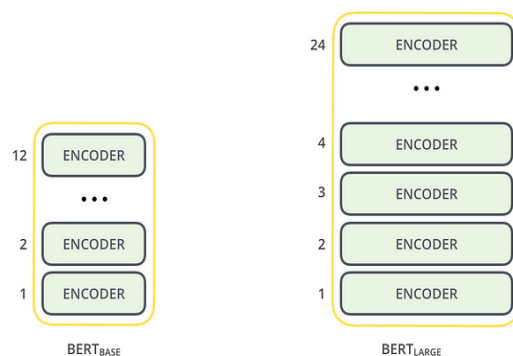
BERT is one of the earliest examples of the encoder only family.

The Architecture

Before we begin grasping parts of the theory, it is important to discuss the architectural properties of BERT. Information about the released models allows us to properly select a specific version of the model during implementation.

Two versions of the BERT Architecture were released in 2018, called BERT Base and BERT LARGE.

The key difference between both of them being the number of stacked encoders that were used. BERT base had 12 encoders stacked on top of each other, meanwhile the large one had 24 stacked encoders.



The embeddings themselves are of size 768 in BERT Base, whereas 1024 for BERT Large, and the Attention Head count is 12 and 16 respectively for Base and Large.

Training And Testing

Similar to Transformer Architecture, the training and testing phases of BERT architecture are different from each other. The reason is a mechanism called Transfer learning, where a model is trained on a specific task, and the weights are reused once more for several other use cases.

This technique of generating “Knowledge” from data and then transferring it to use for your specific use case was developed after to combat the large training times required to train models that encode Language.

In NLP this “Knowledge” about a language is generated using Language Modelling which is the task of predicting the next word based on a given context.

The weights trained after performing the Language Modelling, are referred to as “knowledge” the transformer gained during training and then this knowledge can be applied to more specific use cases, by attaching different heads to the model architecture.

For example: you can do a Reddit Post category classification task using a labelled corpus by adding a softmax layer at the end of the weights trained after language modelling. The weights will update according to the use case of Reddit Post classification and then the model will be able to classify reddit posts and attach flairs if you want to.

Training

The Architecture BERT is trained slightly differently than the typical transformer architecture. Instead of the Language being modelled by a single objective, the “Knowledge” we talked about is generated using two objectives.

- Next Sentence Prediction
- Masked Language Modelling

In BERT two sentences are provided as input at an instance, One is normal target sentence to be swallowed by the encoder, and the other is a sentence that may or may not be adjacent to the currently sent input sentence.

If the second sentence sent is an appropriate option to be the Next sentence to our input sentence, then in that case the model must return 1. If it isn't then 0.

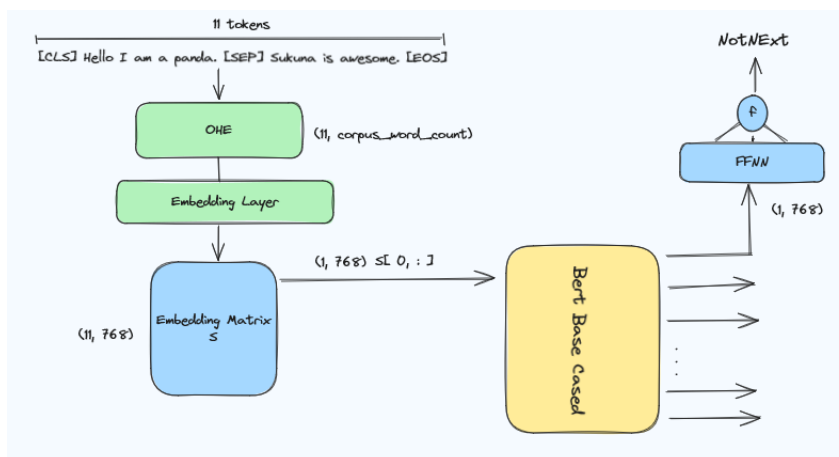
The second task which is word prediction, randomly masks 15% of the words in the input sentence received in batch. These masks are to be processed and predicted correctly by the transformer.

For both these tasks two separate heads are used to get the solution. The results are compared with the labels and then the weights are updated.

Because MLM Masks are not really blocking the words in future or past, but choosing words at random, information from both sides is used to predict the masked word. This is why the architecture is called “Bidirectional”.

The first input provided to the BERT encoder is the [CLS] token. The output produced from this token is used as a Next sentence Prediction output. This first vector is passed through a softmax layer and used to predict whether the pair of sentences is simultaneous.

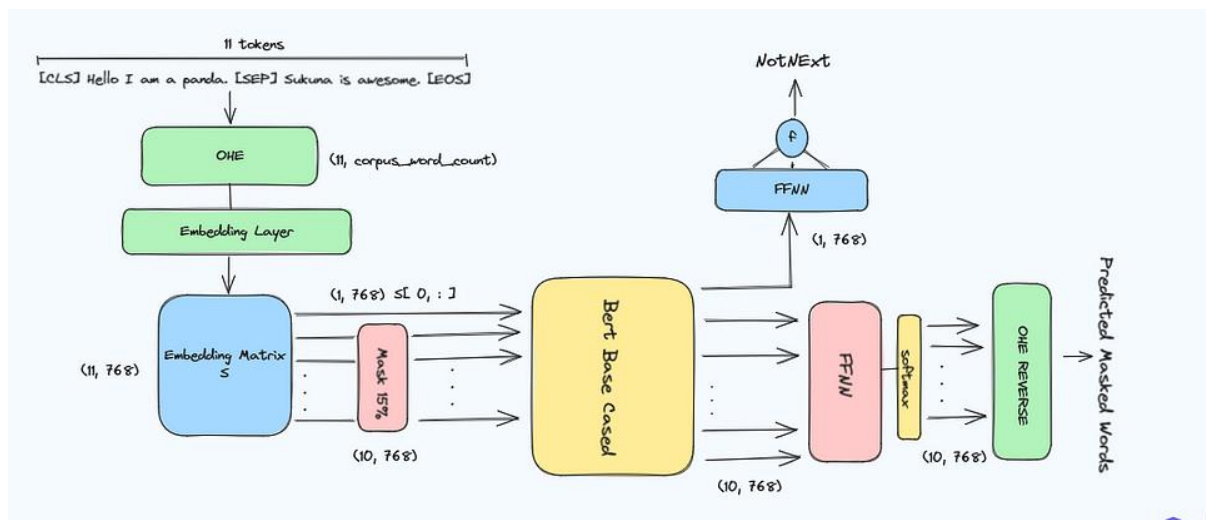
The rest of the vectors are passed through softmax layers as well to produce the final output words.



As you can see in the Image, The sentence is pre-processed to a certain format and paired with another sentence from the corpus. The other sentence will either be next to it or not.

The status for whether or not the sentence is next or not will be provided by the user as label. The words will be used by the BERT transformer all at once and the embedding returned from processing the CLS token is going to be used for predicting if the sentences are neighbours or not.

This preprocessing will of course be influenced by other word embeddings from the input, which isn't shown here in the input because that involves showing you a different step as well, which is involved in Word Prediction.



The rest of the vectors are randomly masked and sent to the transformer, to be encoded accordingly. The masking here is shown outside the Transformer however, we know from the mathematical equation in the previous article that this masking happens inside the transformer.

However, this way is more visually intuitive. The indices of the masked inputs are stored persistently stored and passed on to the other encoders to keep masking at the attention layer of all the encoders. The Z vector outputs are sent through a feed forward layer just like a normal transformer and then the decoder.

```
from transformers import AutoModel, AutoTokenizer
import torch
checkpoint = "bert-base-cased"

sentence = "[CLS] Hello I am a panda. [SEP] Sukuna is awesome. [EOS]"
model = AutoModel.from_pretrained(checkpoint)
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

input_ids = tokenizer.encode(sentence, add_special_tokens=True)
input_ids = torch.tensor([input_ids])

print(input_ids.shape)
```



```
with torch.no_grad():
    last_hidden_states = model(input_ids)[0]

print(last_hidden_states.shape)
```

The output to this is:

```
torch.Size([1, 21])
torch.Size([1, 21, 768])
```

The sentence was broken down into more tokens than expected. But that's probably okay. Now let's see how the masking and attention happens.

Next is random masking before applying before applying Attention.

```
words = embeddings.shape[0]

mask_indices = torch.randint(0, words, (int(0.15 * words),))
mask = torch.zeros((words, words))

for index in mask_indices:
    mask[index, mask_indices] = -torch.inf

S = (torch.mm(Q, torch.transpose(K, 1, 0)) + mask) / torch.sqrt(torch.tensor(d_model))
Z = torch.mm(nn.functional.softmax(S, dim=1), V)
```

The output Z can then be normalized and then processed again by the Feed forward Neural Network. Here the mask is going to ensure that the masked words are not getting any correlation from other masked words.

An unmasked word is allowed to get correlation from masked words. The reason being that this is a distinctive training process where we are making the model adjust the weights by hinting at word should exist in the mask by taking inference from the future and past words but not the masked words themselves.

Consider the sentence:

“I [MASK] sakuna king of darkness and [MASK] indeed yes.”

Let's ignore the extra tokens for now for simplicity ([CLS], [SEP])

Out matrix for masking should look as follows:

		I	[MASK]	Sakuna	King	of	darkness	and	[MASK]	indeed	yes
I	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]
[MASK]	[0.,	-inf,	0.,	0.,	0.,	0.,	0.,	-inf,	0.,	0.]
Sakuna	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]
King	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]
of	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]
darkness	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]
and	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]
[MASK]	[0.,	-inf,	0.,	0.,	0.,	0.,	0.,	-inf,	0.,	0.]
indeed	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]
yes	[0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.]

```
ffnn = nn.Linear(in_features=d_model, out_features=d_model)
corpus_size = 4000
output_softmax = nn.Linear(in_features=d_model, out_features=corpus_size)
vector = ffnn(Z)
output = nn.functional.softmax(output_softmax(vector), dim=0)
torch.argmax(output[mask_indices], dim=1)

=====

output: tensor([ 532,  532, 2315])
```

At the end it gives out word input_ids of the tokens that you would want the words to be.

This is the general flow of the training phase of the BERT transformer.

During Testing the weights essentially do the work of taking your input and encoding it into a hidden state that can be stored and used in any task of your desire. For example if it is a classification task you can attach a softmax layer and pass the FIRST hidden state that is produced by the Encoder and used for NSP.

During testing, since the inputs aren't masked, you can use the multiple vectors that come out as the output of vectors to do Named Entity recognition using your own customized definition of Entities. The possibilities are nearly endless.

The reason these "Word encodings" are so useful is because they rightfully capture the meaning of a sentence based on a given context of a dataset.

If the dataset is large enough it will be able to capture the contextual meaning of a word in the general sense.

For example, if you provide BERT with corpus from the Game of Thrones universe then for the model, the word Dragon will be a "real" creature, because it is a real creature in the Game of Thrones universe.

However, if you give it a large general real-world dataset, in which Game of thrones is simply a subset of data, the Model will treat the word “Dragon” as a fictional creature.

The more you increase the size of your data, the wider your context gets, and the “truer” underlying meaning of the word, the encoder will capture.

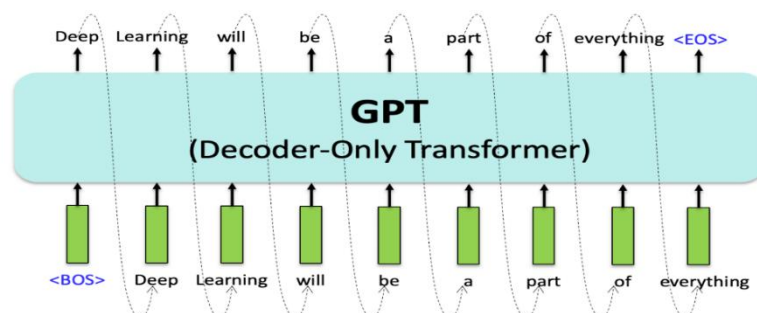
III. DECODER ONLY LLMS : GPT, LLAMA

Introduction

Large-language models (LLMs) have gained tons of popularity lately with the releases of ChatGPT, GPT-4, Bard, and more. All these LLMs are based on the transformer neural network architecture. The transformer architecture was first introduced in the paper "[Attention is All You Need](#)" by Google Brain in 2017. LLMs/GPT models use a variant of this architecture called de' decoder-only transformer'.

The most popular variety of transformers are currently these GPT models. The only purpose of these models is to receive a prompt (an input) and predict the next token/word that comes after this input. Nothing more, nothing less.

Note: Not all large-language models use a transformer architecture. However, models such as GPT-3, ChatGPT, GPT-4 & LaMDa use the (decoder-only) transformer architecture.



Overview of the (decoder-only) Transformer model

It is key first to understand the input and output of a transformer:

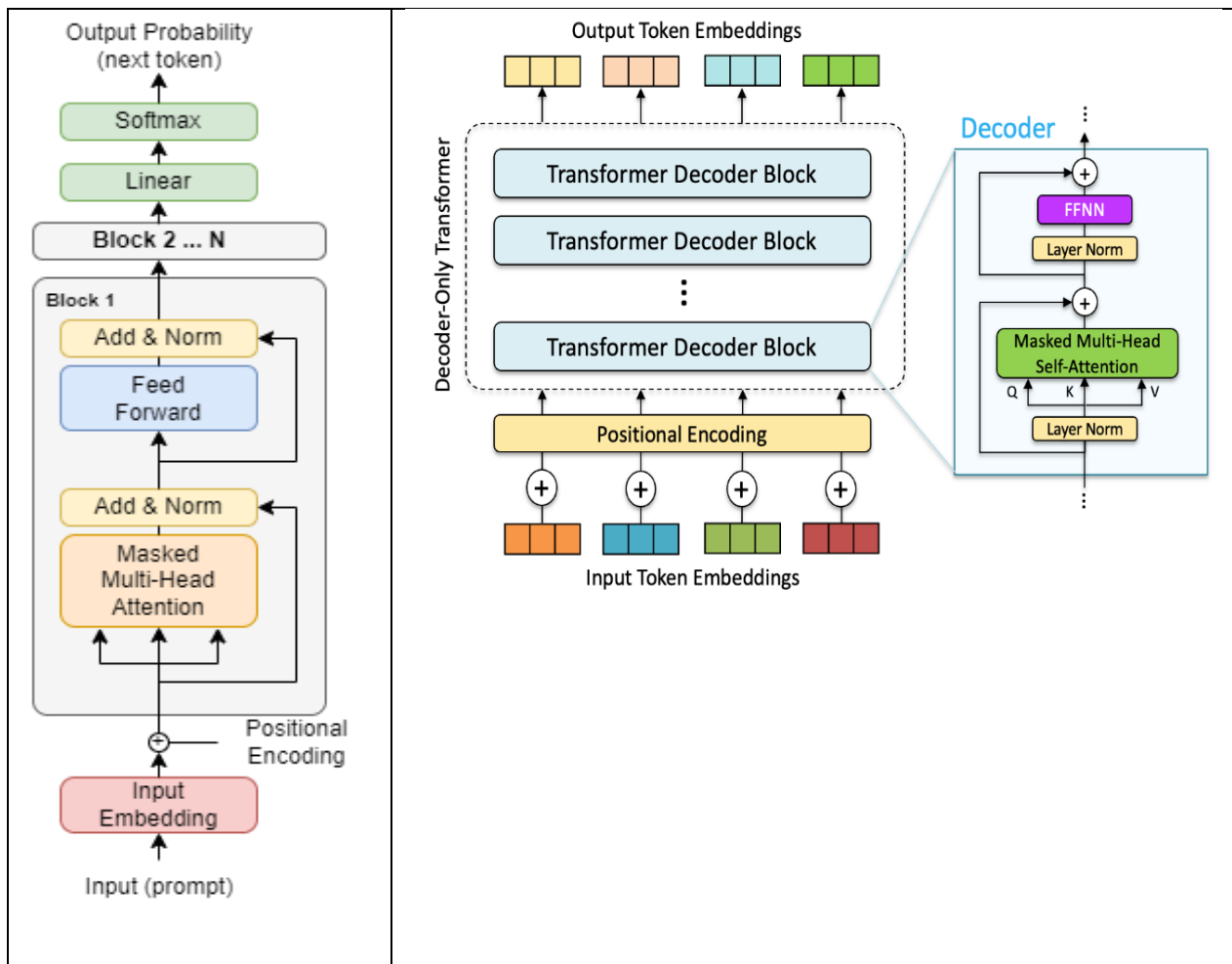
1. **The input** is a prompt (often referred to as context) fed into the transformer as a whole. There is no recurrence.
2. **The output** depends on the goal of the model. For GPT models, the output is a probability distribution of the next token/word that comes after the prompt. It outputs **one** prediction for the complete input.

Next, it is essential to understand the key components that make up the decoder-only transformer architecture:

1. **The embedding:** the input of the transformer model is a prompt. This prompt needs to be embedded into something that the model can use.

2. **The block(s):** This is the main source of complexity. Each block contains a masked multi-head attention submodule, a feedforward network, and several layer normalization operations. *Blocks are put in sequence* to make the model deeper.
3. **The output:** the output of the last block is fed through one more linear layer to obtain the final output of the model (a classification, a next word/token etc.)

The following visualization gives an overview of the transformer architecture.



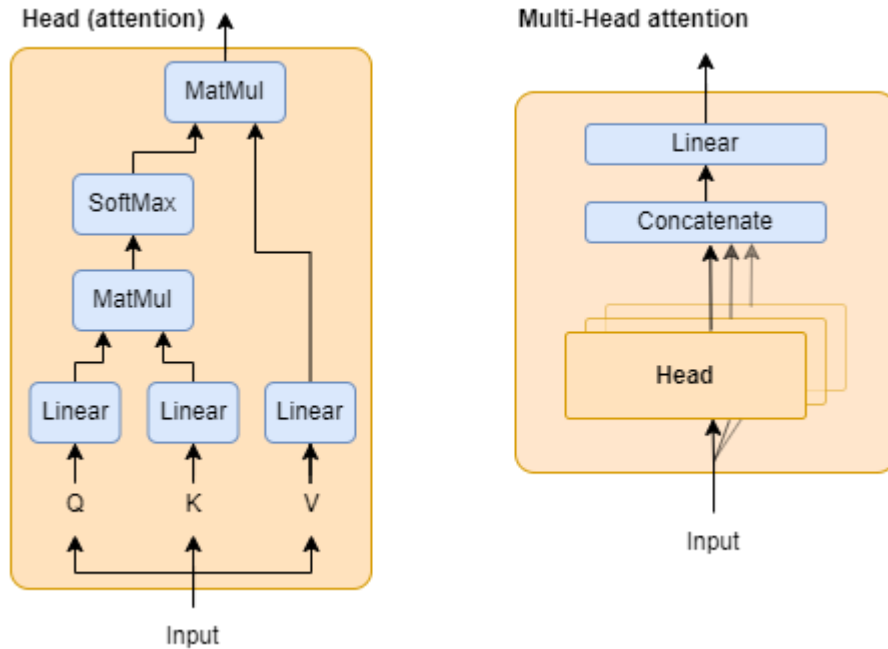
Self-Attention Mechanism

Self-attention makes the transformer powerful. The intuition of self-attention is that the mechanism allows the model to focus on (attend to) the most relevant parts of the input. A single self-attention mechanism is called a **head**.

The head works as follows. First, the input is fed into three separate linear layers. Two of those (the queries Q and the keys K) are multiplied, scaled, and turned into a probability distribution using a softmax activation function. Think of this probability distribution as describing which indices matter most for the output (i.e. which words in the prompt matter for the next word to be predicted). Finally, the output is multiplied with values V . This thus

gives $V * \text{importance}$ of each of the tokens in V . A key observation is that the *learnable parameters* in the head are the three linear layers.

The following figure gives an overview of the operations done in a head and an overview of how multi-head attention works.



Multi-Head Attention

Multi-head attention is nothing more than several individual heads stacked on top of one another. The input to all heads is equivalent. However, each head has its own weights. After forwarding the input through all the heads, the output of the heads is concatenated and passed through a linear layer which brings the dimensionality back to the dimension of the initial input.

Each head processes different aspects of the input, enhancing the model's ability to generate nuanced and contextually appropriate outputs. This mechanism can be described by the following equations:

$$\text{MaskedMultihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}^O$$

Where

$$\text{head}_i = \text{MaskedAttention}(\mathbf{Q} \mathbf{W}_i^Q, \mathbf{K} \mathbf{W}_i^K, \mathbf{V} \mathbf{W}_i^V)$$

By using multiple heads, the model can attend to different parts of the input sequence simultaneously, capturing complex dependencies between tokens while ensuring that each token only attends to past tokens.

Masked Self-Attention

In the decoder-only transformer, masked self-attention is nothing more than sequence padding. The 'masking' term is a left-over of the original encoder-decoder transformer module in which the encoder could see the whole (original language) sentence, and the decoder could only see the first part of the sentence which was already translated. As such, they called it 'masking'.

A critical feature of decoder-only transformers is the use of masked multi-head self-attention. This mechanism ensures that each token can only attend to previous tokens in the sequence, thus preventing any information leakage from future tokens. The mask matrix \mathbf{M} is defined such that:

$$M_{ij} = \begin{cases} -\infty & \text{if } j > i \\ 0 & \text{otherwise} \end{cases}$$

By incorporating this mask into the scaled dot-product attention calculation, the attention scores for future tokens are effectively rendered negligible after applying the softmax function, ensuring that only past tokens influence the current prediction. The masked self-attention mechanism can be described by the following equations:

$$\text{MaskedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}$$

By adding \mathbf{M} to the scaled dot-product, the attention scores for future tokens become very small. After applying the softmax function, these scores are effectively zero, ensuring that the model does not attend to future tokens.

Block

Each block contains a multi-head attention submodule, a feedforward network, 2 layer-normalization operations, and 2 skip connections.

The feedforward network is simply a multi-layer perceptron. In the original paper, the proposed feedforward module consisted of (1) a fully connected layer; (2) a ReLU activation; (3) another fully connected layer; and (4) a dropout layer.

The 'add & norm' blocks get the output from the multi-head attention/feedforward submodule and add it to the input into those modules. After that, a layer normalization operation is performed. Adding the input and output of a submodule together is known as a **skip-connection**. As blocks can be put in sequence, the skip connections help tremendously reduce the problem of vanishing or exploding gradients. In other words, skip connections are necessary to ensure proper backpropagation of the gradients.

Positional Embedding

Transformers take in a complete prompt at once (in contrast to RNNs) and embed this as one big Tensor. As such, transformers do not know which word is at what position in the sentence.

This is problematic as the following two sentences mean entirely different things, only dependent on the order of the words:

<i>The boy chased the bird with a butterfly net.</i>
<i>The bird chased the boy with a butterfly net.</i>

To that end, a positional embedding is added to allow the model to deduce which word is where. Positional embeddings can be learned using any embedding layer from your favourite AI library. However, the original authors proposed a much more complicated method, which does not require learning any parameters. Please find an elaborate explanation (not mine) on the original positional embedding [here](#).

Output

After the prompt is forwarded through all the blocks sequentially, the output is forwarded through one final linear layer. This final linear layer maps the output of the model back to the size of the 'vocabulary'. I.e. if you want to predict the next letter in a message, it would map to 26 (letters) + additional stuff (such as .,-!? etc.).

The output of the model is a probability distribution. For GPT models, the output is the probability of each token being the next token in the sequence.

Training

Basic training

The basic training process consists of [self-supervised learning](#). Simply put, you gather lots of text, strip the last word from that text, feed it as input into the transformer, check if the prediction matches the word, you cut off and backpropagate the error.

Every text/sentence/book/webpage can be separated into several samples.

<i>sample = [</i>
<i>["This"],</i>
<i>["This", "is"],</i>
<i>["This", "is", "a"]] # padding is added until the max-sequence length is reached.</i>
<i>targets = ["is", "a", "sample"]</i>

Fine-tuning / transfer-learning

After the first stage of training is completed, the model is now a large-language model. As in, it can predict the next word based on a context. However, through fine-tuning/transfer-learning the model can be adapted to better suit the needs of the final application.

One of the key reasons why ChatGPT & GPT-4 seem so ridiculously impressive is because of this second stage of training. In this stage, the following process is executed *many* times:

1. The model is given a prompt and generates different answers
2. The different answers are ranked by a human from best to worst.
3. The scores of the different answers are backpropagated.

However, transformer models can also be used for different tasks than language generation. They can, for example, be used for sentiment analysis. After doing the basic training, a transformer can be fine-tuned for sentiment analysis by removing the outgoing linear layer and replacing it with a different layer suitable for the task to be executed. Consequently, it can be trained in a supervised fashion on a custom (sentiment analysis) dataset.

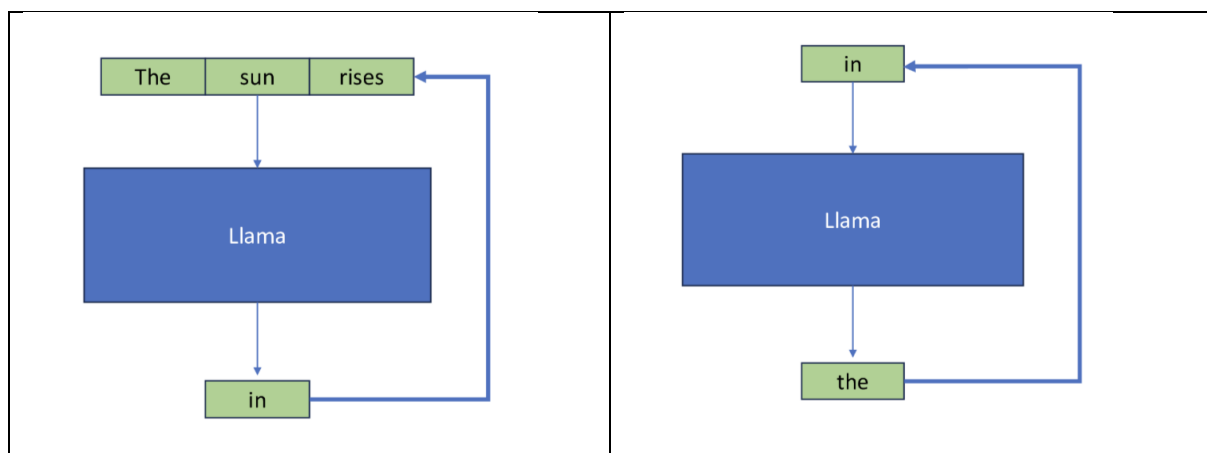
Inference (answer generation)

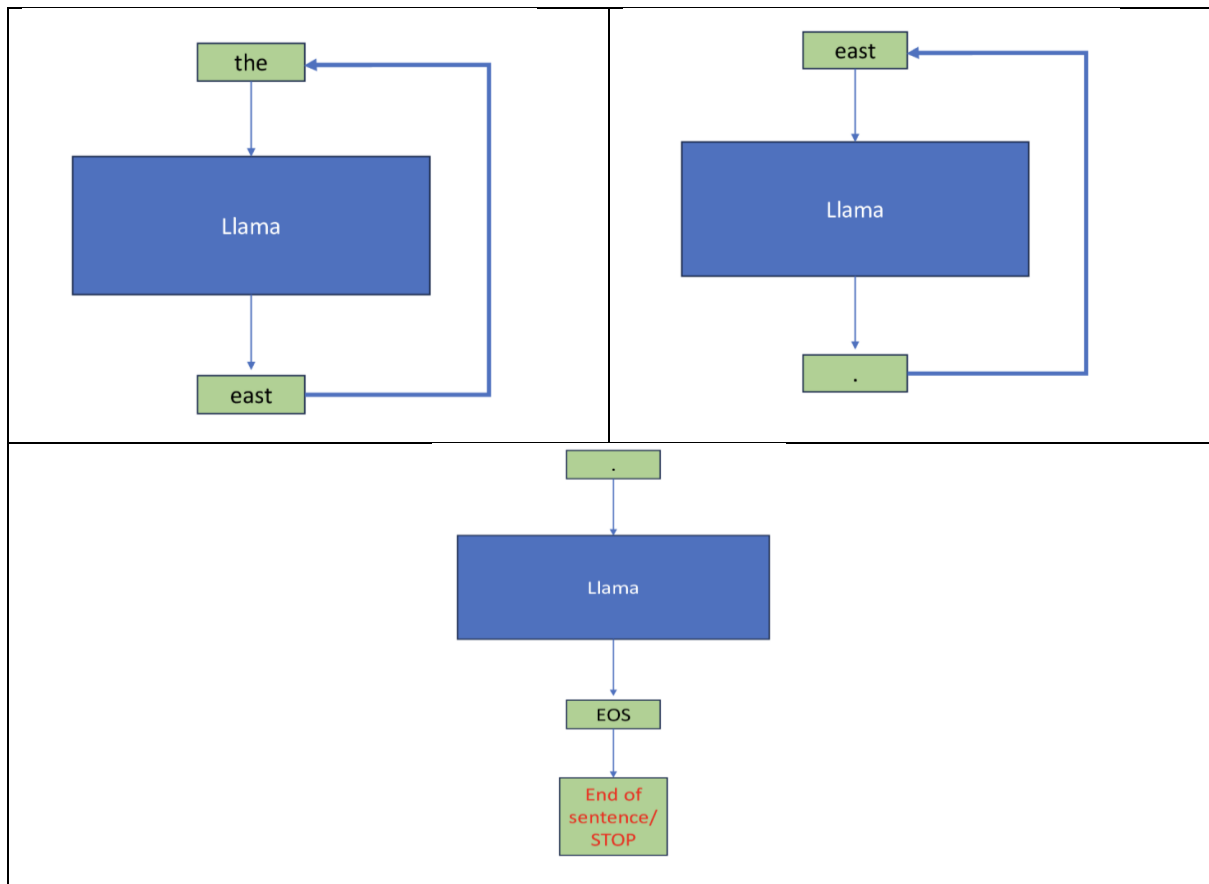
Doing inference with a transformer is just like training. You insert a prompt and out comes the next word/classification/other.

For GPT models, this means that the prompt is extended one word at a time. You insert the prompt, and out comes the first word of the answer. The first word of the answer is now added to the prompt, creating a new, slightly different prompt. This prompt is again forwarded through the model, giving the prediction of a new word.

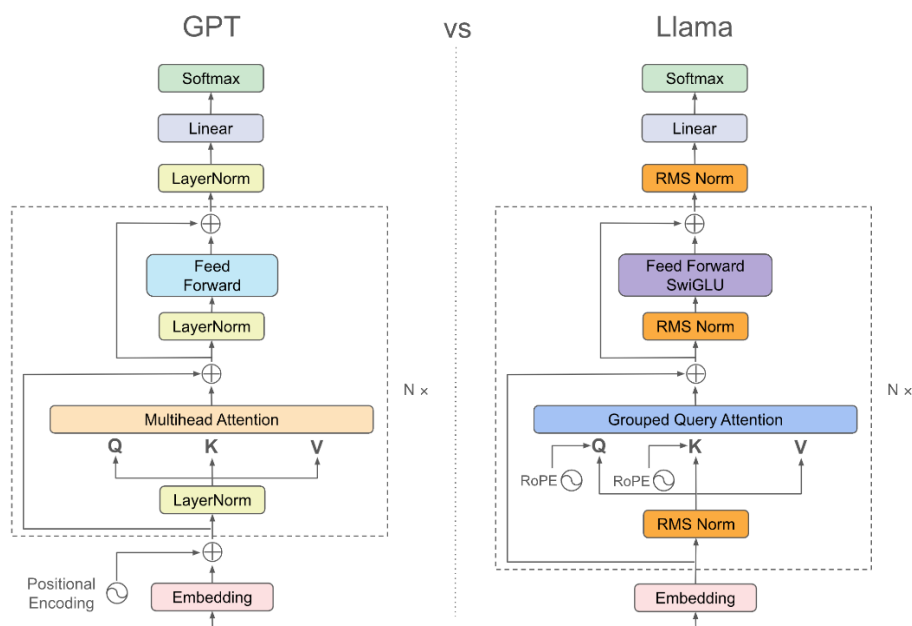
As the output is the probability for each token to be the next one, you can do several things during inference. For one, you can **sample** from the probability distribution. This induces some randomness into the algorithm. You can also take the token that has the highest probability; then, the model becomes deterministic.

Meta's LLaMA Series





Meta's LLaMA series, which stands for **L**arge **L**anguage **M**odel **M**eta **A**I, is a collection of state-of-the-art autoregressive language models developed by Meta AI. Launched in February 2023, the LLaMA models are designed to provide high-performance language understanding and generation across various applications while maintaining a focus on accessibility and efficiency. The series has evolved through several iterations, each introducing enhancements in architecture, training methodologies, and capabilities.



The LLaMA model series can be viewed as a variation of the decoder-only Transformer architecture found in GPT models, with several significant modifications.

1. **RMS Normalization** is applied after input embeddings, rather than after each block.
2. **Rotary Positional Encodings** are used instead of traditional positional encoding and are applied only to the Q and K matrices.
3. **KV Cache and Grouped Multi-Query Attention** are employed to enhance self-attention performance and efficiency.
4. The Feed Forward layer uses **SwiGLU** instead of ReLU.

Overview of the LLaMA Model Series:

- **LLaMA (Initial Release) — February 24, 2023:** The original LLaMA model was introduced in February 2023, featuring a range of sizes from 7 billion to 65 billion parameters. This model was trained on publicly available datasets, including CommonCrawl, C4, GitHub, Wikipedia, and more, enabling it to perform exceptionally well across various natural language processing (NLP) benchmarks. The LLaMA model architecture is based on the transformer framework and is optimized for efficiency, allowing researchers and developers to leverage powerful AI capabilities without requiring extensive computational resources.
- **LLaMA 2 — July 18, 2023:** On July 2023, Meta released LLaMA 2, which included three model sizes: 7B, 13B, and 70B parameters. This iteration utilized 40% more training data than its predecessor and introduced models fine-tuned for chat applications. Notably, LLaMA 2 was made more accessible for commercial use by providing open weights under specific conditions. The performance of the LLaMA 2 models surpassed that of many larger models like GPT-3 in various NLP tasks.
- **LLaMA 3 — April 18, 2024:** The LLaMA 3 series debuted in April 2024 with two primary sizes: 8B and 70B parameters. These models were pre-trained on approximately 15 trillion tokens gathered from publicly available sources. The introduction of LLaMA 3 marked significant improvements in reasoning capabilities, code generation, and instruction-following tasks compared to previous versions. Meta's testing indicated that the LLaMA 3 models outperformed competitors like Gemini Pro and Claude Sonnet on numerous benchmarks.
- **LLaMA 3.1 — June 2024:** Following the success of LLaMA 3, Meta released LLaMA 3.1, which further enhanced the model's performance by optimizing training procedures and improving alignment and diversity in responses. LLaMA 3.1 maintains the same model sizes of 8B, 70B, and introduces a larger size of 405B parameters, featuring further enhancements in efficiency, performance, and multimodal capabilities, including text, image, and potentially audio inputs.

- **LLaMA 3.2 — September 27, 2024:** The latest iteration, LLaMA 3.2, introduced smaller models specifically designed for low-latency applications while also incorporating multimodal capabilities for visual understanding tasks. This release includes models such as LLaMA-3.2–1B-Instruct and LLaMA-3.2–90B-Vision-Instruct, enabling enterprises to leverage these models for tasks like document parsing and product description generation. The multimodal aspect allows for enhanced
- Overall, the LLaMA series exemplifies Meta’s commitment to advancing AI technology through innovative design and accessibility while driving forward the open-source movement in artificial intelligence development.

Conclusion

The evolution of Large Language Models (LLMs), led by OpenAI’s GPT series, has revolutionized natural language processing (NLP) and AI. Decoder-only transformer architectures and self-supervised pretraining have enabled models like ChatGPT to achieve human-like text generation and understanding.

Key players like Anthropic, Google, Meta, Mistral AI, and Alibaba have further advanced LLMs. Anthropic’s Claude series focuses on safety, Google’s Gemini on multimodal capabilities, Meta’s LLaMA on accessibility, Mistral AI on open-source principles, and Alibaba’s Qwen on performance and multilingual support.

The landscape of LLMs is shifting towards greater openness, with models like Mistral 7B and LLaMA 2 providing access to weights, code, and data, promoting transparency and ethical AI practices. Meanwhile, closed models like GPT-4 and Google Bard rely on proprietary services. The open-source community’s progress in alignment techniques is narrowing the gap between open and closed models, as seen with the LLaMA3.1–405B model.

IV. THE MIXTURE-OF-EXPERTS (MOE)

Training large language models (LLMs) often demands significant computational resources, which can be a barrier for many organizations and researchers.

The concept MoE originated from the 1991 paper *Adaptive Mixture of Local Experts*. Since then, MoEs have been employed in multi-trillion parameter models, such as the 1.6 trillion parameter open-sourced Switch Transformers.

The Mixture of Experts (MoE) model is a class of transformer models. MoEs, unlike traditional dense models, utilize a “sparse” approach where only a subset of the model’s components (the “experts”) are used for each input. This setup allows for more efficient pretraining and faster inference while managing a larger model size.

In MoEs, each expert is a neural network, typically a feed-forward network (FFN), and a gate network or router determines which tokens are sent to which expert. The experts specialize in different aspects of the input data, enabling the model to handle a wider range of tasks more efficiently.

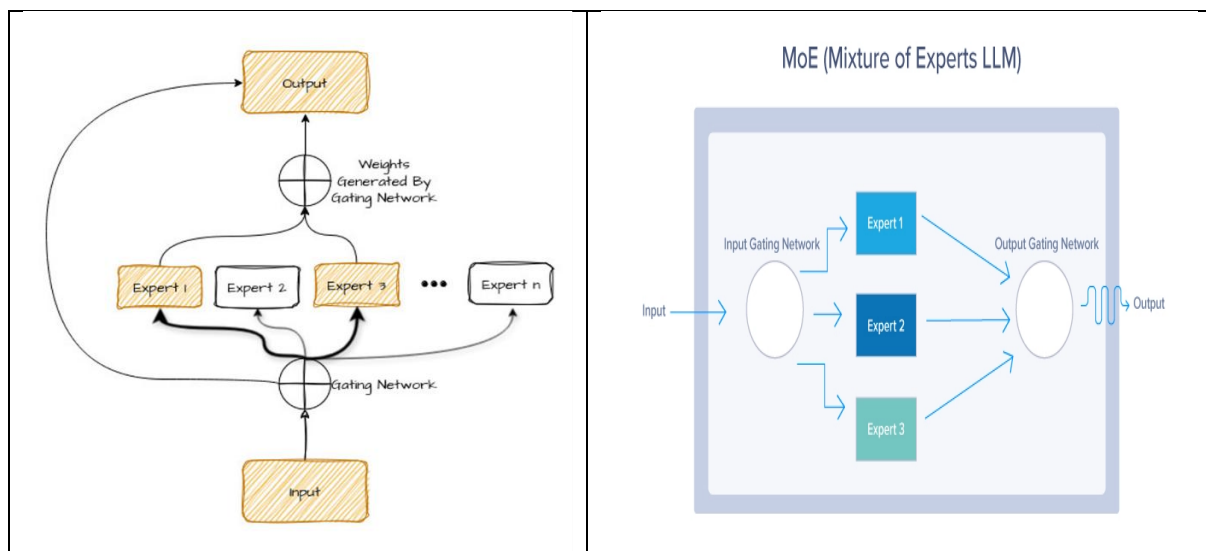
Imagine an AI model as a team of specialists, each with their own unique expertise. A mixture of experts (MoE) model operates on this principle by dividing a complex task among smaller, specialized networks known as “experts.”

An analogy for understanding MoEs is to consider a hospital with various specialized departments (experts). Each patient (input token) is directed to the appropriate department by the reception (router) based on their symptoms (data characteristics). Just as not all departments are involved in treating every patient, not all experts in an MoE are used for every input.

Each expert focuses on a specific aspect of the problem, enabling the model to address the task more efficiently and accurately. It’s similar to having a doctor for medical issues, a mechanic for car problems, and a chef for cooking—each expert handles what they do best.

By collaborating, these specialists can solve a broader range of problems more effectively than a single generalist.

Let’s take a look at the diagram below.



Let’s break down the components of this diagram:

- **Input:** This is the problem or data you want the AI to handle.
- **Experts:** These are smaller AI models, each trained to be really good at a specific part of the overall problem. Think of them like the different specialists on your team.
- **Gating network:** This is like a manager who decides which expert is best suited for each part of the problem. It looks at the input and figures out who should work on what.

- **Output:** This is the final answer or solution that the AI model produces after the experts have done their work.

MoE Gating Network

The gating network acts as the decision-maker or controller within the MoE model. It evaluates incoming tasks and determines which expert is suited to handle them. This decision is typically based on learned weights, which are adjusted over time through training, further improving its ability to match tasks with experts. The gating network can employ various strategies, from probabilistic methods where soft assignments are tasked to multiple experts, to deterministic methods that route each task to a single expert.

MoE Experts

Each expert in the MoE model represents a smaller neural network, machine learning model, or LLM optimized for a specific subset of the problem domain. For example, in Mistral, different experts might specialize in understanding certain languages, dialects, or even types of queries. The specialization ensures each expert is proficient in its niche, which, when combined with the contributions of other experts, will lead to superior performance across a wide array of tasks.

MoE Loss Function

Although not considered a main component of the MoE architecture, the loss function plays a pivotal role in the future performance of the model, as it's designed to optimize both the individual experts and the gating network.

It typically combines the losses computed for each expert which are weighted by the probability or significance assigned to them by the gating network. This helps to fine-tune the experts for their specific tasks while adjusting the gating network to improve routing accuracy.

How Mixture of Experts (MoE) Works

MoE operates in two stages:

1. The training phase
2. The Inference phase

Training phase

Similar to other machine learning models, MoE begins by training on a dataset. However, the training process is not applied to the entire model but is instead conducted on its components individually.

Expert training

Each component of an MoE framework undergoes training on a specific subset of data or tasks. The aim is to enable each component to focus on a particular aspect of the broader problem.

This focus is achieved by providing each component with data relevant to its assigned task. For instance, in a language processing task, one component might concentrate on syntax while another on semantics.

The training for each component follows a standard neural network training process, where the model learns to minimize the loss function for its specific data subset.

Gating network training

The gating network is tasked with learning to select the most suitable expert for a given input.

During the training of the gating network, is trained alongside the expert networks. It receives the same input as the experts and learns to predict a probability distribution over the experts. This distribution indicates which expert is best suited to handle the current input.

The gating network is typically trained using optimization methods that include both the accuracy of the gating network and the performance of the selected experts.

Joint training

In the joint training phase, the entire MoE system, which includes both the expert models and the gating network, is trained together.

This strategy ensures that both the gating network and the experts are optimized to work in harmony. The loss function in joint training combines the losses from the individual experts and the gating network, encouraging a collaborative optimization approach.

The combined loss gradients are then propagated through both the gating network and the expert models, facilitating updates that improve the overall performance of the MoE system.

Inference phase

Inference involves generating outputs by combining context from gating networks with outputs from experts. In MoE, this process is designed to keep inference costs minimal.

Input routing

In the context of MoE, the role of the gating network is pivotal in deciding which models should process a specific input.

Upon receiving an input, the gating network assesses it and creates a probability distribution across all the models. This distribution then directs the input to the most suitable models, leveraging the patterns learned during the training phase. This ensures that the right expertise is applied to each task, optimizing the decision-making process.

Expert selection

Only a select few models, usually one or a few, are chosen to process each input. This selection is determined by the probabilities assigned by the gating network.

Choosing a limited number of models for each input helps in the efficient use of computational resources while still benefiting from the specialized knowledge within the MoE framework.

The output from the gating network ensures that the chosen models are the most appropriate for handling the input, thereby improving the system's overall efficiency and performance.

Output combination

The last step in the inference process involves merging the outputs from the selected models.

This merging is often achieved through weighted averaging, where the weights reflect the probabilities assigned by the gating network. In certain scenarios, alternative methods like voting or learned combination techniques might be employed to merge the expert outputs. The aim is to integrate the varied insights from the selected models into a unified and accurate final prediction, thereby leveraging the strengths of the MoE architecture.

With the rapid advancement of technology, there is an increasing need for fast, efficient, and optimized techniques to handle large models. MoE is emerging as a promising solution in this regard.

MoE's Advantages Over Traditional Models and its Challenges

Advantages

The mixture of Expert (MoE) architecture offers several advantages over traditional deep learning models.

Enhanced scalability: MoE models scale easily to billions or even trillions of parameters due to sparse activation, which reduces the need for massive computational power.

Increased flexibility: One unique benefit of MoE is that new experts can be added to the existing model without retraining the entire system. This adaptability allows the model to readily accommodate new tasks and domains.

Efficiency: Since MoE activates only the most relevant experts for each input, it can handle diverse tasks more efficiently than traditional models. This makes it faster and more accurate, as experts can focus on what they do best.

Parallel processing: Experts can work independently, which allows for efficient parallel processing. This approach can cause faster training and inference times.

Challenges

Despite the above advantages, MoE models also have certain challenges and limitations.

Training complexity: Training MoE models can be challenging, particularly in managing the gating network and balancing the contributions of individual experts. It's important to ensure that the gating network learns to assign appropriate weights to experts effectively, which prevents overfitting or underutilizing specific experts.

Communication costs: MoE models require significant infrastructure resources during training and inference due to managing multiple experts and the gating mechanism. Additionally, when deployed at scale, especially across various devices or distributed systems, the communication overhead becomes a major challenge. Coordinating and synchronizing

outputs from various experts on different servers can lead to increased latency and computational load.

Expert Capacity: Thresholds are set on the number of inputs each expert can process simultaneously to prevent overloading specific experts and ensure a balanced workload. A common approach is to use top-2 routing with a 1.25 capacity factor, meaning that two experts are selected per input, and each expert processes 1.25 times its usual capacity. This strategy also assigns one expert per core, optimizing performance and resource management.

Interpretability: Opacity is already a notable problem in AI, including for leading LLMs. Mixture-of-experts (MoE) models can make this even harder because they are more complicated. Instead of just looking at how one model makes decisions, we must figure out how different experts and the gating system work together. This extra complexity can make it much harder to understand why the model made a certain choice.

Applications of MoE

Mixture-of-Experts is already being used in a wide range of applications.

Natural language processing: Mixture-of-experts (MoE) models are great for language tasks like translation, sentiment analysis, and answering questions because they can assign each task to specialists. For example, it's been reported that OpenAI's GPT-4 large language model uses an MoE setup with 16 experts, although OpenAI hasn't officially confirmed its design details. Another example is Microsoft's translation API, Z-code. The MoE architecture in Z-code allows the model to support a massive scale of model parameters while still using the same computing power.

Computer vision: Google's V-MoEs, a sparse architecture based on Vision Transformers (ViT), show how effective MoE can be for computer vision tasks. MoE models can help with image processing by giving different tasks to specialized experts. For example, one expert might focus on certain types of objects, specific visual features, or other parts of an image.

Multimodal learning: MoE can combine data from multiple sources, such as text, images, and audio, into a model. This makes MoE ideal for applications like multimodal search or content recommendation, where data from different modalities needs to be integrated.

Future Directions for MoE

In the coming years, research on Mixture-of-Experts (MoE) will focus on making the models more efficient and easier to understand. This includes improving how experts work together and finding better ways to assign tasks to the right experts.

Further scaling: MoE models can be scaled to even larger sizes while minimizing computational costs. This includes optimizing training and inference phases to handle the increasing size of experts and data. Techniques such as distributed computing are being explored to spread tasks across multiple machines more efficiently, reducing bottlenecks and speeding up the model's operation.

Innovative routing mechanisms: Another area of research focuses on developing more efficient routing strategies. While existing methods like top-k routing are commonly used, advanced techniques such as Expert Choice Routing aim to improve task assignment to experts. This can lead to more efficient load balancing and improved task-to-expert matching, ensuring the model performs optimally under different conditions.

Real-World applications: MoE models have great potential in real-world applications like healthcare and autonomous systems due to their ability to handle complex tasks. In healthcare, they could be used for personalized treatment plans, while autonomous systems could use them for tasks like object recognition and decision-making.

V. SIGNIFICANCE OF DECODER ONLY LLMS OVER ENCODER ONLY LLMS

Decoder-only and encoder-only language models (LLMs) have distinct architectures and strengths, making them suitable for different types of tasks. Understanding their significance requires exploring their design and applications:

Decoder-Only LLMs

- **Architecture:** These models, like GPT (Generative Pre-trained Transformer), consist of a stack of transformer decoders. They are trained to predict the next token in a sequence, making them autoregressive.
- **Strengths:**
 1. **Text Generation:** Ideal for generating coherent and contextually relevant text due to their autoregressive nature. They excel at tasks like story generation, summarization, and dialogue systems.
 2. **Flexibility:** Can perform a wide range of natural language understanding (NLU) and generation tasks using prompting techniques.
 3. **Few-shot and Zero-shot Learning:** Pre-training on large datasets equips them with strong generalization abilities, allowing them to perform tasks with minimal or no task-specific fine-tuning.
 4. **Scalability:** They scale well in performance with an increase in model size and training data, as demonstrated by models like GPT-4.

Encoder-Only LLMs

- **Architecture:** Encoder-only models, like BERT (Bidirectional Encoder Representations from Transformers), consist of transformer encoders and are designed for bidirectional context understanding. They predict masked tokens, enabling deep contextual representation.
- **Strengths:**

1. **Understanding Context:** Superior at extracting meaning from text, making them ideal for tasks like sentiment analysis, named entity recognition, and question answering.
2. **Pretraining for Fine-tuning:** Their representations are tailored for downstream tasks, where fine-tuning yields highly accurate results.
3. **Parallelization:** They can process sequences more efficiently for tasks requiring bidirectional context, unlike autoregressive decoders that rely on sequential processing.

Significance of Decoder-Only Models Over Encoder-Only Models

Decoder-only language models (LLMs) offer significant advantages over encoder-only models, particularly in text generation tasks. Their architecture, which focuses solely on the decoding process, enhances efficiency by reducing computational resources and accelerating processing time, making them ideal for applications requiring rapid text generation. Additionally, decoder-only LLMs excel in producing fluent and coherent text, as they are specifically designed to generate well-structured outputs without the limitations faced by encoder-only models. This fluency is crucial in sequence-to-sequence tasks, such as machine translation and text summarization, where maintaining context and relevance is essential. Furthermore, in language translation, decoder-only LLMs demonstrate flexibility and efficiency, providing a viable alternative to traditional encoder-decoder architectures. Overall, the advantages of decoder-only LLMs make them a preferred choice for various natural language processing applications.

1. Generative Capabilities:

- Decoder-only models are inherently better at text generation, whereas encoder-only models are not designed for this.
- Applications like creative writing, conversational agents, and code generation favor decoder-only models.

2. Few-shot and Zero-shot Tasks:

- Decoder-only models can generalize and perform tasks with minimal examples by leveraging in-context learning. Encoder-only models require explicit fine-tuning for task-specific adaptation.

3. Unified Workflow:

- Decoder-only models handle both NLU and generation tasks using the same architecture and interface, making them versatile for multi-purpose applications.

4. Scalability and Adaptability:

- Large-scale decoder-only models tend to exhibit emergent behaviors, handling complex reasoning or multi-modal inputs more effectively.

Limitations and Complementarity

- Decoder-only models struggle with tasks requiring deep bidirectional context. For example, understanding intricate dependencies across large spans of text is more natural for encoder-only models.
- Many applications use **encoder-decoder models** (like T5) to combine the strengths of both, excelling at tasks requiring both comprehension and generation.

In summary, decoder-only LLMs are significant for their flexibility, generalization, and suitability for creative and generative tasks. However, their utility should be considered alongside encoder-only and encoder-decoder models for a comprehensive approach to NLP challenges.

Performance in Speech Tasks

Decoder-only LLMs have demonstrated superior performance in speech-to-text translation, achieving state-of-the-art results on benchmarks like CoVoST 2 and FLEURS.

They can directly consume encoded speech representations, streamlining the translation process and enhancing efficiency.

Efficiency and Inference Speed

Implementing online knowledge distillation techniques allows decoder-only models to reduce character error rates significantly while maintaining lower inference costs compared to encoder-decoder architectures(Lee & Seo, 2024).

The FocusLLM framework enhances the context length capabilities of decoder-only models, improving their performance on long-context tasks without substantial resource demands(Li et al., 2024).

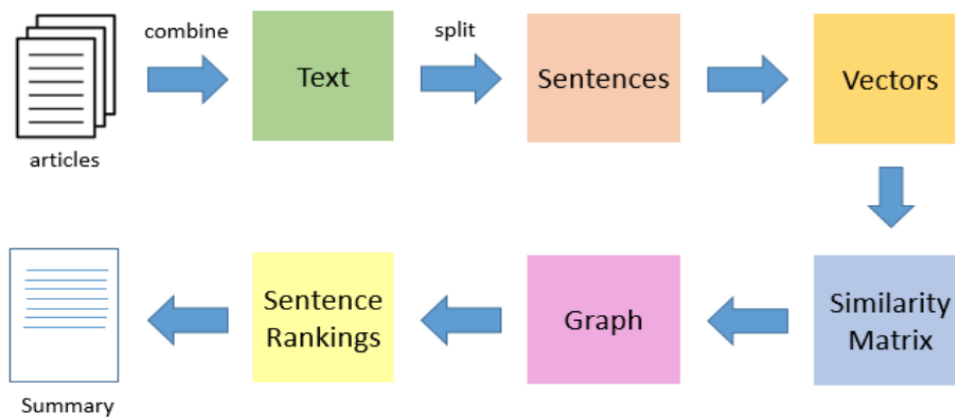
While decoder-only models excel in generation tasks, encoder-only models may still be preferred in scenarios requiring detailed contextual understanding, such as complex comprehension tasks. This highlights the complementary nature of both architectures in the broader landscape of LLM applications.

VI. SUMMARIZATION IN NLP

The volume of written information has increased exponentially over the years thanks to the rise of the internet and the increasing popularity of social media. As a result, the ability to efficiently break down large chunks of text into smaller and easily understandable sentences or paragraphs has become crucial for many organizations.

Unfortunately, it's quite tedious and time-consuming to manually summarize lengthy texts. You also risk leaving out important details when sifting through sensitive documents. This is where text summarization comes in handy.

Below is an in-depth review of what text summarization is, how it works, various techniques used, and its most common use cases.



What is text summarization for NLP?

Text summarization is a subset of Natural Language Processing (NLP) that uses advanced algorithms and machine learning models to analyze and break down lengthy texts into smaller digestible paragraphs or sentences. This procedure extracts the most valuable information from a particular text without altering its original meaning. It can be applied in a wide variety of domains, including academia, business, and news.

Text summarization not only reduces the time and effort required to read and understand lengthy texts, but it also ensures the accuracy and completeness of a summary. It is often useful when the text has many raw facts that can be filtered from it. This makes NLP text summarization a valuable tool when it comes to summarizing technical documents, financial materials, and sensitive legal texts.

The goal of text summarization is to take a long text as input and generate a short version with all the relevant facts. This is a much more complicated task than the previous ones since it requires the model to generate coherent text.

In what should be a familiar pattern by now, we can instantiate a summarization pipeline as follows:

```

summarizer = pipeline("summarization")
outputs = summarizer(text, max_length=45, clean_up_tokenization_spaces=True)
print(outputs[0]['summary_text'])

```

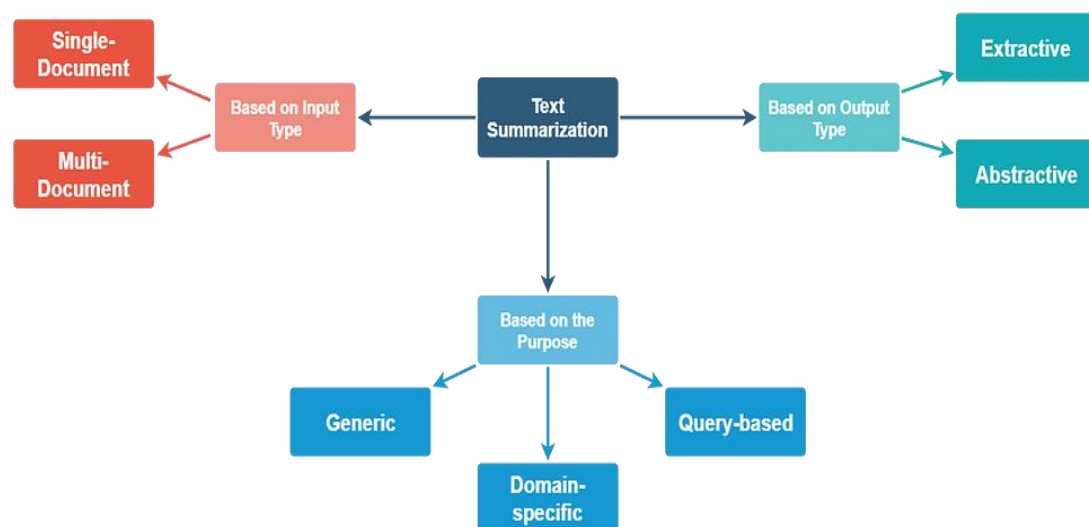
Bumblebee ordered an Optimus Prime action figure from your online store in Germany. Unfortunately when I opened the package I discovered to my horror that I had been sent an action figure of Megatron instead.

This summary isn't too bad! Although parts of the original text have been copied, the model was able to capture the essence of the problem and correctly identify that "Bumblebee" (which

appeared at the end) was the author of the complaint. In this example you can also see that we passed some keyword arguments like `max_length` and `clean_up_tokenization_spaces` to the pipeline; these allow us to tweak the outputs at runtime. But what happens when you get feedback that is in a language you don't understand? You could use Google Translate, or you can use your very own transformer to translate it for you!

NLP text summarization techniques

Here are the most common types of NLP text summarization techniques:



Input-based NLP text summarization:

One of the most common ways of classifying NLP text summarization is based on the input. In this case, text summarization can be from a single document or multiple documents. Single-document summarization basically refers to an NLP text summarization technique whereby the input is from a single document. Most of the early text summarization systems commonly dealt with single-document summarization. [1]

On the other hand, multi-document summarization involves an input of multiple documents. Multi-document NLP text summarization is a bit more challenging compared to single-document summarization because it requires the ability to analyze and understand the underlying relationship between multiple documents/texts and extract a summary of the essential information from them.

Output-based NLP text summarization:

Another great way of classifying NLP summarization is on the basis of the output or format of the final summary. There are two types of output-based text summarization, namely extractive and abstractive text summarization.

Extractive text summarization:

Extraction summarization is an NLP summarization technique that involves selecting and isolating essential information from a pre-existing text or document and compressing the

information to form a summary. You can think of this summarization technique as a highlighter that allows you to see the main points in a given text.



In machine learning, extractive summarization is based on features such as word frequency, sentence centrality, text ranking, and sentence length. This procedure also relies on machine learning models and graph-based algorithms to weigh key sections of a given text and use the results to generate a summary or shortened form of the original text. Today, most NLP text summarization techniques are extractive in nature.

Abstractive text summarization:

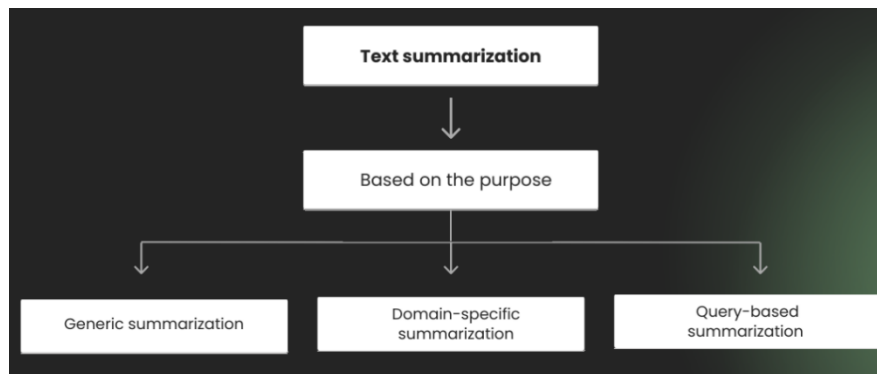
Abstractive text summarization is an NLP summarization technique that uses Natural Language Generation (NLG) to generate a summary that is a new and unique representation of the original text. This technique relies on advanced machine learning and natural language processing to understand the meaning and context of the original text and then generates a summary that captures the main ideas.

Basically, abstractive text summarization is quite similar to how humans summarize texts. In our minds, we usually create a phonological representation of the text and pick terms from our daily vocabulary that align with the semantics of the original text; then we generate a meaningful summary. That said, it's possible that a summary generated using abstract text summarization contains phrases that aren't present in the original text.



Purpose-based NLP text summarization:

On the basis of the purpose or goal of the summary, NLP text summarization can be classified into the following:



Generic summarization:

Generic text summarization is a summarization technique that provides an overview of the main points or ideas in a particular text without making any assumptions about the content. This technique simply condenses or summarizes the original text and treats all inputs as homogenous. Most of the work done in text summarization revolves around generic summarization.

Domain-specific summarization:

This is a type of summarization technique that uses domain-specific knowledge to generate a summary that is tailored to the domain of the original text. For example, machine learning models can be trained using medical science terminology so that they can generate more accurate summaries of texts and documents in this particular field.

In most cases, domain-specific summarization techniques are used to summarize legal documents, technical reports, and other specialized texts.

Query-based summarization:

The primary focus of this type of NLP text summarization is to generate a summary that answers specific natural language questions about a particular text. This is a similar technique to the one used to generate search results on popular search engines such as Google and Bing.

Every time you type a question on a search engine like Google, it returns a selection of websites or articles that answer your question. The search results usually contain a summary of an article that directly answers your question or is relevant to what you're searching for.

Common use cases for NLP text summarization

Financial Research: Summarizing financial reports and statistics to identify trends and inform investment decisions.

Media Monitoring: Breaking down publications into concise summaries to stay informed about industry events.

Chapters for YouTube Videos and Podcasts: Creating chapters or sections for content to enhance viewer and listener experience.

Email Thread Summarization: Summarizing email conversations to quickly identify key points.

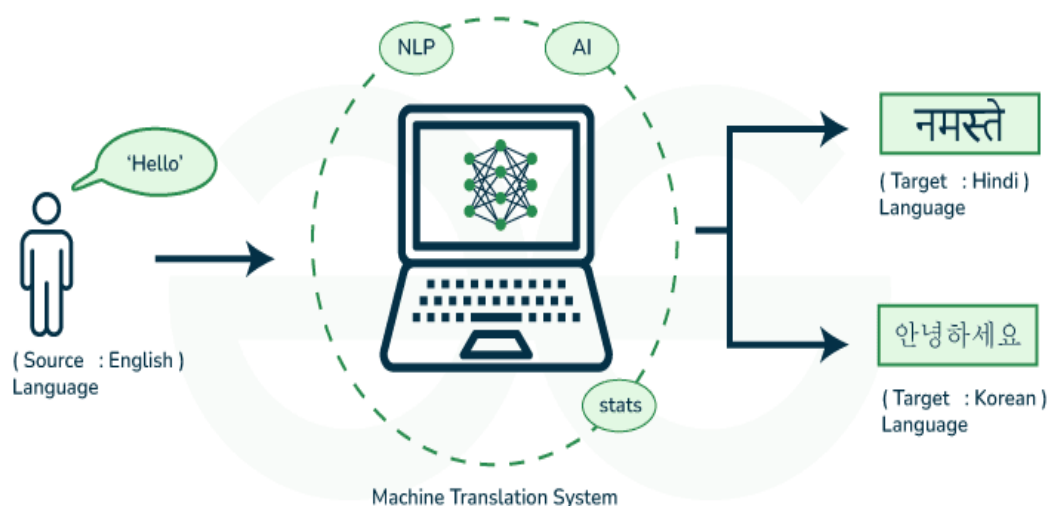
SEO: Analyzing search engine results to optimize web content for higher rankings.

Customer Feedback and Reviews: Summarizing feedback to identify common issues and improve products.

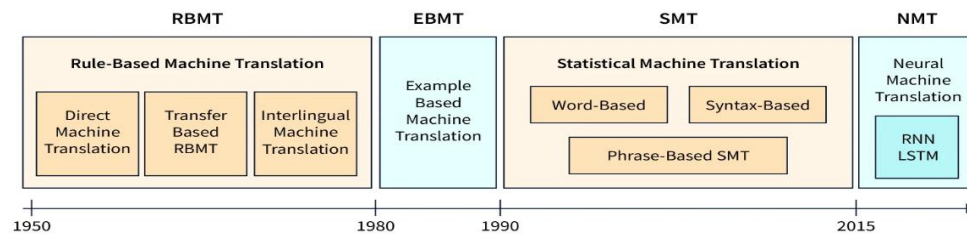
Legal Document Analysis: Summarizing legal documents to understand key points without reading the entire text.

VII. TRANSLATION IN NLP

With thousands of different languages used all over the world, translation is one of the most exciting applications of natural language processing technology. NLP has made a great deal of progress in the area of machine translation, making it possible for computers to translate text and speech from one language to another with high accuracy.



Evolution of Machine Translation



How Language Translation has changed over Time?

Machine learning and artificial intelligence (AI) have significantly changed the domain of language translation, making it more accurate and efficient.

One of the most notable developments in this field is the use of neural networks, which has led to a significant increase in translation quality. With the help of natural language processing (NLP) algorithms, machines can now accurately figure out what words and phrases in a text mean and translate them more precisely.

Recent progress has also been helped by the availability of large amounts of data and the development of new techniques, such as transfer learning.

How can Language Translation solve Real-World Problems?

Language translation can help solve a wide range of problems in the real world, such as making it easier for people from different cultures and countries to talk to each other, making international trade easier, and improving education and health care.

For businesses, language translation can help them reach new markets and customers. It can also help them figure out what their target audience wants and needs, which is important for making good marketing plans.

In the healthcare field, language translation can help doctors and nurses communicate clearly with patients who speak different languages. This can help doctors make better diagnoses and treatments and make patients happier.

In a similar way, language translation can solve problems in many other industries. If you think of anything, feel free to tell us in the comments.

Like summarization, translation is a task where the output consists of generated text. Let's use a translation pipeline to translate an English text to German

```
translator = pipeline("translation_en_to_de" model="Helsinki-NLP/opus-mt-en-de")
outputs = translator(text_clean_up_tokenization_spaces=True min_length=100)
print(outputs[0]['translation_text'])
```

Sehr geehrter Amazon, letzte Woche habe ich eine Optimus Prime Action Figur aus Ihrem Online-Shop in Deutschland bestellt. Leider, als ich das Paket öffnete, entdeckte ich zu meinem Entsetzen, dass ich stattdessen eine Action Figur von Megatron geschickt worden war! Als lebenslanger Feind der Decepticons, Ich hoffe, Sie können mein Dilemma verstehen. Um das Problem zu lösen, Ich fordere einen Austausch von Megatron für die Optimus Prime Figur habe ich bestellt. Anbei sind Kopien meiner Aufzeichnungen über diesen Kauf. Ich erwarte, bald von Ihnen zu hören. Aufrichtig, Bumblebee.

Again, the model produced a very good translation that correctly uses German's formal pronouns, like "Ihrem" and "Sie." Here we've also shown how you can override the default model in the pipeline to pick the best one for your application—and you can find models for thousands of language pairs on the Hugging Face Hub. Before we take a step back and look at the whole Hugging Face ecosystem, let's examine one last application.

Different Types of Machine Translation in NLP

Machine translation methods are mainly classified into rule-based and corpus-based methods during the developments from the 1950s till now. Recently neural network-based models are primarily employed for translation tasks using NLP.

Let us discuss and learn the classification and methods involved in each type of model.

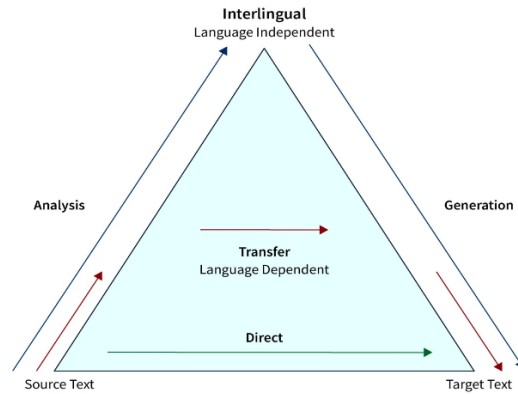
Rule-based Machine Translation or RBMT:

Also called knowledge-based machine translation, these are the earliest set of classical methods used for machine translation.

These translation systems are mainly based on linguistic information about the source and target languages that are derived from dictionaries and grammar covering the characteristic elements of each language separately.

Once we have input sentences in some source languages, RBMT systems generally generate the translation to output language based on the morphological, syntactic, and semantic analysis of both the source and the target languages involved in the translation tasks.

The sub-approaches under RBMT systems are the direct machine translation approach, the interlingual machine translation approach, and the transfer-based machine translation approach.



Corpus-based Machine Translation Approach or CBMT:

Also called data driven machine translation, these methods majorly overcome the problem of knowledge acquisition problem of rule-based machine translation approaches. The Corpus-based machine translation methods can handle linguistic issues and generally find it difficult to translate cultural items and concepts.

- Corpus-Based Machine Translation is a set of methods that uses a bilingual parallel corpus to obtain knowledge for new incoming translation.
- This approach uses a large amount of raw data in the form of parallel corpora. This raw data contains text and their translations. These corpora are used for acquiring translation knowledge.
- Corpus-based approaches are classified into two sub-approaches: Statistical Machine Translation and the Example-based Machine Translation approach.

Example-based Machine Translation or EBMT:

Example-based machine translation systems are trained from bilingual parallel corpora where the sentence pairs contain sentences in one language with their translations into another.

- The foundational idea of example-based machine translation is the concept of translation by analogy.
 - The principle of translation by analogy is encoded to the example-based machine translation through the example translations that are used to train the system.
- The bilingual corpus with parallel texts serves as the main knowledge for an Example-based machine translation system in which translation by analogy.
 - A set of sentences in the source language are given in the example-based machine translation system, and corresponding translations of each sentence in the target language with point-to-point mapping are also provided as input.
 - These examples are used to translate similar types of sentences of the source language to the target language.

- Example acquisition, example base, and management, example application, and synthesis are the four main tasks to be performed in EBMT methods.

Sample example for EBMT-based approach:

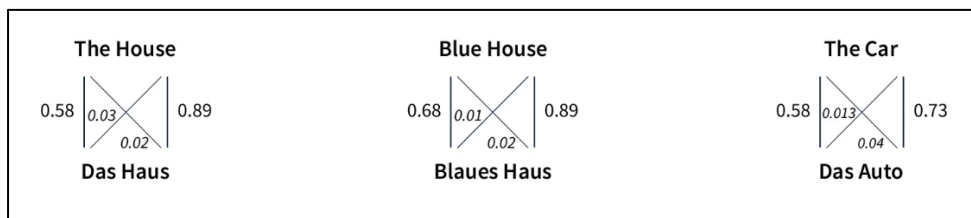
- Let us say we want to translate a simple sentence — I'm going to the playground., and let's say that we have already translated another similar sentence — I'm going to the movies. — and we can find the word “movies” in the dictionary.
- All we need is to figure out the difference between the two sentences, and then translate the missing word in the new language.
- The more examples we have, the better the translation will be.

Example-based machine translation systems paved the path further to statistical-based machine translation systems.

Statistical Machine Translation or SMT

Statistical machine translation (SMT) models are generated with the use of statistical models whose parameters are derived from the analysis of bilingual text corpora. The ideas for SMT come from information theory.

- The principle theory of SMT-based models is that they rely on the Bayes Theorem to perform the analysis of parallel texts.
 - The central idea is that every sentence in one language is a possible translation of any sentence in the other language, but the most appropriate is the translation that is assigned the highest probability by the system.
- The SMT methods need millions of sentences in both the source and target languages for analysis to collect the relevant statistics for each word.
 - In the early set of models, the abstracts of the European Parliament and the `United Nations Security Council meetings, which were available in the languages of all member countries, were taken for getting the required probabilities.
- SMT-based methods are further classified into Word-based SMT, Phrase-based SMT, and Syntax based SMT approaches.



Issues with SMT-based Approaches

- Sentence Alignment: As we are using two corpora to match sentences, many times we will encounter single sentences in the source language, which can be found translated into several sentences in the target language.

- Example: When translating from English to German, the sentence "Raghu does not live here," the word "does" doesn't have a clear alignment in the translated sentence "Raghu wohnt hier nicht."
- Statistical Anomalies: Real-world training sets may override translations like proper nouns.
 - Example: "I took the train to Berlin" gets mistranslated as "I took the train to Paris" due to an abundance of "train to Paris" in the training set generally.
- Idioms: Depending on the corpora used, idioms may not translate idiomatically into the target language.
 - For example, using Canadian Hansard as the bilingual corpus, "hear" may almost invariably be translated to "Bravo!" since in Parliament, "Hear, Hear!" becomes "Bravo!".
- Different Word Orders: Since the word orders in languages differ, Statistical machine translation methods do not work well between languages that have significantly different word orders, for example Japanese and European languages.
- Corpus Creation: This can be very costly for users with limited resources, and sometimes the results are also unexpected where the superficial fluency can be deceiving.
- Data Dilution: This is one of the most common anomalies caused when attempting to construct a new statistical model to represent a distinct terminology for a specific corporate brand or domain.

Hybrid Machine Translation or HMT

- Hybrid Machine Translation approaches take advantage of both statistical and rule-based translation methodologies which were proven to have better efficiency in the area of machine translation systems using NLP.
 - These techniques are better than the previous methods and have more power, flexibility, and control in translation tasks in general.
 - Most governmental and industrial-based machine translation systems use the hybrid-based approach to develop translation from source to target language, which is based on both rules and statistics.
- Hybrid-based approaches combine the best of both worlds such that they can be used in several different ways.
 - In cases where translations are performed in the first stage using a rule-based approach, they can be followed by adjusting or correcting the output using statistical information.
 - On the other way, rules are used to pre-process the input data as well as post-process the statistical output of a statistical-based translation system.

Neural Machine Translation or NMT

Neural Machine Translation methods in NLP employ artificial intelligence techniques and, in particular, rely upon neural network models which were originally based on the human brain to build machine learning models with the end goal of translation and also further learning languages, and improving the performance of the machine translation systems constantly.

- Neural Machine Translation systems do not need any specific requirements that are regular to other machine translation systems like statistical-based methods in general.
- Neural Machine Translation works by incorporating training data, which can be generic or custom depending on the user's needs.
 - Generic Data: Includes the total of all the data learned from translations performed over time by the machine translation engines employed in the entire realm of translation.
 - This data also serves as a generalized translation tool for various applications, including text, voice, and documents.
 - Custom or Specialized Data: These are the sets of training data fed to machine translation tasks to build specialization in a subject matter.
 - Subjects include engineering, design, programming, or any discipline with its specialized glossaries and dictionaries.

Advantages of NMT Models:

1. NMT has achieved the state of the art performance in large-scale translation tasks in many languages like English to French, English to German etc.
2. NMT requires minimal domain knowledge and is conceptually very simple and powerful.
3. NMT has a small memory footprint as it does not store gigantic phrase tables and language models.
4. NMT also can generalize well to very long word sentences.

VIII. NAMED ENTITY RECOGNITION (NER)

Named Entity Recognition (NER) is a process in Natural Language Processing (NLP) that identifies and classifies key elements (named entities) in text into predefined categories such as the names of persons, organizations, locations, dates, etc.

In today's world, where data is abundant and comes in various forms, extracting meaningful information from text (called "unstructured") has become paramount. Information Extraction (IE) is a field that deals with the automatic extraction of structured information from unstructured sources, such as natural language text.

Process of NER:

1. Preprocessing:

It involves cleaning the text, removing unnecessary elements (like HTML tags, if any) and sometimes converting the entire text into a uniform case (usually lowercase) to standardize the input data.

2. Tokenization:

The text is split into sentences and then into words or tokens.

3. Part-of-Speech Tagging:

Each word or token is labeled with a part of speech (noun, verb, adjective, etc.), based on its definition and its context in the sentence.

4. Entity Detection:

It can be approached in several ways including rule-based methods, machine learning models or a combination of both. Rule-based methods use predefined patterns and dictionaries while machine learning models especially those based on deep learning learn from annotated training data.

5. Entity Classification:

Once potential entities are detected, these are classified into predefined categories.

6. Post-processing:

It might involve normalization as standardizing entity representations e.g., dates into a common format and disambiguation as ensuring entities are correctly identified in terms of their real-world counterparts.

Named Entity Recognition Predicting the sentiment of customer feedback is a good first step, but you often want to know if the feedback was about a particular item or service. In NLP, real-world objects like products, places, and people are called named entities, and extracting them from text is called named entity recognition (NER). We can apply NER by loading the corresponding pipeline and feeding our customer review to it:

```
ner_tagger = pipeline("ner", aggregation_strategy="simple")
outputs = ner_tagger(text)
pd.DataFrame(outputs)
```

	entity_group	score	word	start	end
0	ORG	0.879010	Amazon	5	11
1	MISC	0.990859	Optimus Prime	36	49
2	LOC	0.999755	Germany	90	97
3	MISC	0.556569	Mega	208	212
4	PER	0.590256	##tron	212	216
5	ORG	0.669692	Decept	253	259
6	MISC	0.498350	##icons	259	264
7	MISC	0.775361	Megatron	350	358
8	MISC	0.987854	Optimus Prime	367	380
9	PER	0.812096	Bumblebee	502	511

You can see that the pipeline detected all the entities and also assigned a category such as ORG (organization), LOC (location), or PER (person) to each of them. Here we used the aggregation strategy argument to group the words according to the model's predictions. For example, the entity "Optimus Prime" is composed of two words, but is assigned a single category: MISC (miscellaneous). The scores tell us how confident the model was about the entities it

identified. We can see that it was least confident about “Decepticons” and the first occurrence of “Megatron”, both of which it failed to group as a single entity.

NER is a crucial component of IE that identifies and classifies entities in text into predefined categories such as persons, organizations, locations, dates, and more. For example, in the sentence “Steve Jobs co-founded Apple Inc. in Cupertino, California,” NER would identify “Steve Jobs” as a person, “Apple Inc.” as an organization, and “Cupertino, California” as a location.

Extracting all the named entities in a text is nice, but sometimes we would like to ask more targeted questions. This is where we can use question answering.

```
import spacy
# Load the pre-trained NER model
nlp = spacy.load("en_core_web_sm")
# Text to process
text = "Steve Jobs co-founded Apple Inc. in Cupertino, California."
# Perform NER
doc = nlp(text)
# Print the entities
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Output:

```
Steve Jobs PERSON
Apple Inc. ORG
Cupertino, California LOC
```

IX. WHAT ARE THE APPLICATIONS OF LLM?

Large Language Models (LLMs) have emerged as a powerful tool in the field of artificial intelligence, offering a wide range of applications across various industries. These advanced models can understand and generate human-like text, enabling them to perform tasks that were once thought to be the exclusive domain of humans. We’ll explore some of the most prominent **LLM applications** and how they are revolutionizing the way we interact with technology.

In general, LLMs should be seen as assistants and not as human replacement. Human review and supervision are essential to getting effective use out of LLMs. In this article, we will analyse the following use cases:

1. **Sentiment analysis**
2. **Chatbot**
3. **Content generation**

4. **Summarization**
5. **Text classification**
6. **Cybersecurity**

An LLM application is a method to improve or innovate work using LLMs. LLMs are particularly adept at Natural Language Processing (NLP), which is a process in which machine learning is used to understand and generate human language. While LLMs are an innovative technology, questions may remain about the scope of their usage in a professional context. We will, with these 7 examples illustrating LLM use-cases, attempt to answer that question.

How is an LLM application built? Due to their large size, LLMs are typically deployed locally or in the cloud and called using an API endpoint. In very simple terms: a user can provide a prompt via an interface which is sent to the model and processed. A response is then returned to the user. This allows LLMs to be easily incorporated into websites or other platforms. We at UbiOps have dedicated guides on how to deploy Mistral-7b, BERT, LLaMa 2 and Falcon if you are interested.

1. Using LLMs for Sentiment analysis

Sentiment analysis designates the process of evaluating text and placing it into a certain emotional category. The most basic emotional categories are positive/negative/neutral, but more complex categorizations exist, ranging from disappointment to annoyance. An LLM which is very well suited to this use case is **BERT**.

What is BERT?

BERT is an encoder-only transformer model developed by Google in 2018. Importantly, it is relatively cheap and can be deployed easily. Here are some of its core characteristics.

Name	Number of parameters	Type	License
BERT	110M, 336M	Encoder-only	apache-2.0

BERT can be used as a language classifier. This is why it is well suited for sentiment analysis. Now let's explore some concrete examples, looking at use cases in **financial services** and **customer service analysis**.

Sentiment analysis use cases:

- **Financial services**

Sentiment analysis models can be used to analyze financial headlines. This can help in predicting financial trends. To this end, a fine-tuned version of BERT named FinBERT is an obvious candidate. FinBERT was fine-tuned on a large financial database, allowing it to effectively analyze whether a financial news headline is positive or negative. Here are two examples:

Prompt	Result
Apple CEO states that the company is struggling.	96.6 % Negative, 2.4% Neutral, 1.0% Positive
Hurricane strikes Ford Headquarters!	72.4% Negative, 25.1% Neutral, 2.5% Positive

- **Customer service analysis**

In customer service, analyzing the large number of issues and concerns raised by customers can be arduous if done manually. However, with LLMs, you can easily analyze customer service requests and classify them based on emotion or urgency. A potential model for this use case is a model based on BERT named roberta-base-go_emotions which classifies text into 28 different emotions. Here are two examples:

Prompt	Result
This product sucks and I want a refund now	Annoyance 56.2%, Anger 27.1%, Disapproval 7.8%, Disappointment 6.8%...
This product worked ok until the screen stopped working.	Neutral 42.7%, Approval 37.1%, Realization 11.6%, Disappointment 7.9%...

2. Using an LLM as a chatbot

The most widely known usage of an LLM is to use it as a chatbot. Chatbots are especially useful for companies wanting to use LLMs as helper tools for internal use, i.e. employees, or external use, i.e. customers. Chatbots can be fine-tuned and refined on internal data, allowing you to create a chatbot specific for your use case. In general, chatbot LLMs are flexible and smart, making them a great helper tool. Currently, the best, most cost-effective pre-trained LLM model series is **Mistral**.

What is Mistral?

Mistral is currently the best bang for your buck LLM, meaning that it is both cheap and effective. The model is extremely knowledgeable and has performed very well on a variety of tests. It's a conversational AI, which means that it is designed to simulate human conversations. It's also completely open source and usable for commercial purposes.

Model series	Number of parameters	Type	License
Mistral	7B, 46.7B	Decoder-only	Apache-2.0

Two ways to use chatbot LLMs such as Mistral are as a **customer service agent** or as a **question answerer**. You can make your chatbot even more relevant by fine-tuning it with internal data or by implementing RAG. These methods can help create a chatbot which is

specifically designed for your use case. Mistral, again, is a good model for fine-tuning given that it has performed well on a variety of general logic and intelligence benchmarks. We wrote a guide to help decide when to fine-tune your LLM where we explain this further.

Chatbot use cases:

- **Customer service agent**

Many companies are now using LLMs as customer service agents. Major banks such as ING are already incorporating chatbots into their customer service environment. They basically help human agents by reducing load, potentially identifying what a customer's issue is and guiding them to the correct contact source — all before human review. Here is an example conversation using Mistral:

```
-> Hello! I am the ExampleBank assistant. Need account info or help with transactions? Just share what you need (e.g., balances, transfers). Remember: no sensitive data. Ask anything else; if I don't know, I'll find someone who does!
```

```
-> My credit card broke!
```

```
-> I'm sorry to hear that your credit card got damaged! Please contact our dedicated support team at 1-800-EXAMPLEBANK (1-800-392-6753) for immediate assistance. They will be able to cancel your current card and arrange an emergency replacement quickly. Stay safe!
```

- **Question answering**

Question answering chatbots, especially when fine-tuned on internal or industry-specific data, can have a variety of applications. For instance, Apple is reported to use a custom chatbot for internal use. In general, a company or industry-specific LLM can be a useful tool to help employees or customers find answers to questions. Here is an example using Mistral:

```
-> Hello! I'm here to help answer any questions you have about Google's products and services. What can I assist you with today?
```

For example, if you need help with a specific product like Gmail or Drive, just let me know and I can provide guidance on how to use its features effectively. If you have general inquiries about company policies or benefits, I can also try my best to provide accurate information.

Please keep in mind that while I strive to be helpful, there may be some limitations to what I can assist with as an AI language model. Let's get started!

```
-> A customer wants to integrate google cloud into his python backend. Is it possible?
```

```
-> Absolutely! Here are some popular Google Cloud libraries for Python:
```

- `google-cloud-storage`: Allows reading/writing data to Google Cloud Storage buckets.
- `google-auth`: Simplifies authentication and authorization across Google APIs.
- `google-api-python-client`: Provides methods for accessing various Google APIs.
- `apache_beam[gcp]`: Lets users write parallelizable, portable data processing pipelines for both batch and streaming data.
- `dialogflow`: Enables building conversational interfaces powered by natural language understanding.

These libraries enable seamless integration between your Python application and different parts of the Google Cloud ecosystem, making development faster and easier. Remember to follow setup instructions in official docs first, including creating project IDs and obtaining necessary keys!

3. Using LLMs for content generation

LLMs are also useful for content generation. Companies, either due to licensing or content creator cost, can use LLMs to generate copyright free content. While this aspect of LLMs is still in its infancy stage, it is progressing rapidly and LLMs can be used to optimize **image creation** and generate **source code**.

For image generation, **diffusion-based models** are the way to go. While diffusion models aren't made for NLP, when used in tandem with an LLM, you can optimize your image generating performance.

What is a diffusion model?

Diffusion models, according to the paper Diffusion Models: A Comprehensive Survey of Methods and Applications, are a “family of probabilistic generative models that progressively destruct data by injecting noise, then learn to reverse this process for sample generation.” Diffusion models are perfectly designed for image creation, and many LLMs have been made to optimize their generative capacity.

Content generation use cases:

- **Image generation**

The most popular image generation model to date is Stable Diffusion. It can generate high quality images very quickly. These don't have any license fees as they are copyright free. Companies wanting to evade licensing costs can use a model such as stable-diffusion-2-1. We have a guide on how to deploy Stable Diffusion on our platform. As mentioned, you can use LLMs to enhance your prompt so it generates a better-quality image. We will be using Magic Prompt to do so, which is based on GPT-2. Here is an example:

Prompt	LLM-enhanced prompt
Quaint train station during autumn	Quaint train station during autumn, highly detailed, digital painting, concept art, sharp focus, illustration, detailed, warm lighting, cozy warm tint, magic the gathering artwork, volumetric lighting, 8k, no gold, no gold colours, art by Akihiko Yoshida, Greg Rutkowski



Example of a stable-diffusion generated image with prompt enhancement using Magic Prompt

- **Code generation**

Code generation LLMs are taking off as well. Tools such as GitHub Copilot and ChatGPT have been designed with code generation in mind, and they have caught on in the developer world. According to a GitHub blog post which surveyed 500 developers from US-based companies with 1000+ employees, around **92% of the developers surveyed were using AI coding tools**. This number is extremely high and could be expected to increase as AI coding tools get smarter and smarter.

The widespread use of AI tools raises some concerns, especially if companies rely on closed-source models. Data privacy concerns have been raised with ChatGPT, and many companies have banned the use of ChatGPT internally. If you want to deploy or use AI models, it is important to make sure that the companies you partner with are GDPR-compliant.

Here is an example of code generation using Mistral:

```
-> Write me a python program that retrieves the weather

-----

To retrieve the weather in Python, you can use an API (Application Programming Interface) such as OpenWeatherMap. Here's a simple example of how to do this:

First, sign up for a free account on <https://openweathermap.org/> and get your API key.

Next, install the `requests` library by running `pip install requests`.

Here is the code:
import requests

def get_weather(city):
    api_key = "your_api_key" # replace with your actual API key
    base_url = "http://api.openweathermap.org/data/2.5/weather?"

    complete_url = f"{base_url}appid={api_key}&q={city}"
    response = requests.get(complete_url)
    data = response.json()

    if data["cod"] != "404":
        main_data = data["main"]
        temperature = main_data["temp"] - 273.15 # convert from Kelvin
```

4. Using LLMs for text classification

Retrieving core elements of a body of text is a useful application for any fields needing. In essence, models which are designed for text classification try to place text into a certain category. In general, *zero-shot classification* is useful for this use case as it is very flexible. BART is a well-suited model for this.

What is BART?

BART is an encoder-decoder model developed by Facebook in 2019. For zero-shot classification, Facebook released bart-large-mnli which was trained on a Natural Language Inference (NLI) dataset. Models trained for NLI can easily be used as zero-shot classifiers.

Model	Number of parameters	Type	License
Bart-large-mnli	407M	Encoder-Decoder	MIT

What is zero-shot classification?

Zero-shot classification is a type of classification which enables a user to prompt a model with custom categories. This means that you can define categories at inference time. Zero-shot classifiers are useful if you envisage calling the model in different ways and using different category lists.

Text classification use cases

- **Search optimization**

Using an LLM can help optimize your searches. Here is a simple example: if you offer a search engine for articles or blog posts, you can use a zero-shot classifier model to retrieve the relevant tags from a search prompt and match it with articles with the same tags. Here is an example of how it could be used:

Prompt	Tags	Result
Make a custom LLM	fine-tuning, language models, training, instruction tuning, business, cars	language models 94.5%, fine-tuning 79.3%, business 78.7%, instruction tuning 36.3%, training 12.6%, cars 0.2%
Iphone crack in screen	urgent, not urgent, phone, tablet, computer, fix, guide	phone 99.7%, fix 59.9%, urgent 59.4%, guide 17.2%, not urgent 2%, computer 0.1%, tablet 0%

BART-large-mnli zero-shot prompting for search optimization

- **Question sorting**

If your platform has some sort of feedback form, sorting through the responses can be done effectively using a zero-shot classifier. For instance, if you have different teams dealing with different issues, a zero-shot classifier could allow you to determine which team is best for answering certain questions. Here are a couple of examples, again using bart-large-mnli:

Prompt	Tags	Result
Scroll bar is too fast on homepage	frontend development, backend development	93.0% frontend development, 7.0% backend development
http request returns 500 error	frontend development, backend development	56.2% backend development, 43.8% frontend development

BART-large-mnli zero-shot prompting for question sorting

While the results here may seem underwhelming, remember that you can fine-tune your LLM or use RAG to make it specific to your company/industry. BART-large-mnli is a general use zero-shot classifier.

5. Using LLMs for summarization

Summarizing text is another task that LLMs perform well. Summarizing involves compressing a lengthy piece of text down to its essentials. BART, being an encoder-decoder model, is built for text comprehension and generation and it is well suited for summarization.

Summarization use cases:

- **Media digestion**

Reading the news every single day, especially if you want to be an informed reader and read a variety of outlets, can be a time-consuming task. News summarization services aim to solve this issue. A good model for this task is bart-large-cnn. A model based on BART and fine-tuned on the CNN/DailyMail database.

Here is an example of its summarization capabilities. Provided bart-large-cnn with a BBC article titled: [*Apple watch: US upholds ban on sale over patent dispute*](#). It summarized the article as follows:

Apple was earlier allowed to sell its Series 9 and Ultra 2 watches while proceedings were ongoing. But a US appeals court has since reversed that decision. It is the latest turn in a dispute between the firm and medical technology company Masimo. Masimo and spin-off Cercacor have accused the iPhone maker of poaching key staff.

- **Academic article summarization**

In research or academia, you can use AI to generate a summary of a lengthy article and quickly check if it contains any relevant information. It can help ascertain whether you should spend the time to read an entire article. An example of a model specifically developed for long form articles or books would be led-large-book-summary, a model based on BART, or bigbird-pegasus-large-pubmed, a model developed by Google based on a unique architecture. Here is an example of how it works: we can give led-large-book-summary the first section of a chapter titled [***“THE EU AND ALGERIA: A temporary marriage of convenience?”***](#), then it responds with a summary:

This chapter discusses the contour of the EU-Algeria relations and its relationship with Algeria. The EU is Algeria's main trading partner and Algerian exports to the EU have grown significantly over the years 2002–2014. However, trade will begin to decline in both 2019 and 2020 due to domestic demand and a subsequent decline in exports due to a gas crisis. Europe has been interested in Algeria's natural gas exports since the outbreak of the War in Ukraine in March 2014. Former Italian Prime Minister Mario Draghi, French Prime Minister Giorgia Meloni, and French President Emmanuel Macron all visited Algeria to urge it to increase gas exports. The author argues that the EU/Algeria relationship will be short lived due to the limited Algerian hydrocarbon export capacity and because of divergence of views on issues important to the Algerian and European governments.

- **Cybersecurity**

LLMs have some unfortunate consequences when it comes to cybersecurity, given that the massive chat models such as GPT-4 and Mistral are trained on large sections of the internet. Through their vast training set, they learned hacking techniques as well as fishing techniques. However, they also possess capabilities when it comes to cyber defense.

- **LLMs for cyber offense**

Following the release of GPT-4, the Institute of Electrical and Electronics Engineers (IEEE) released a research paper [documenting the ways GPT-3 could be used for cyberwarfare](#). According to this paper, ChatGPT (when prompted using jailbreaking techniques) can give detailed information on how to perform attacks using:

- SQL injection
- Phishing/social engineering
- Ransomware
- Spyware
- Trojans
- Polymorphic malware.

Given how eloquent LLMs can be when writing an email, detecting phishing attacks will become increasingly difficult as they increasingly resemble a legitimate source. Furthermore, LLMs can generate attacks far faster than humans, potentially increasing the rate of attacks. Let's take a look now at the ways LLMs can be used for cyber defense.

- **LLMs for cyber defense**

According to the same paper, GPT-3/4 can be used to:

- Read server logs and potentially detect vulnerabilities
- Keep you up to date on the cybersecurity industry, detailing new threats and malware that others have identified

- Detect vulnerabilities in code or generate secure code
- Detect unusual activity or patterns
- Identify malicious code

Therefore, while the potential LLMs have for cyber offense is scary, its potential for cyber defence is also powerful.

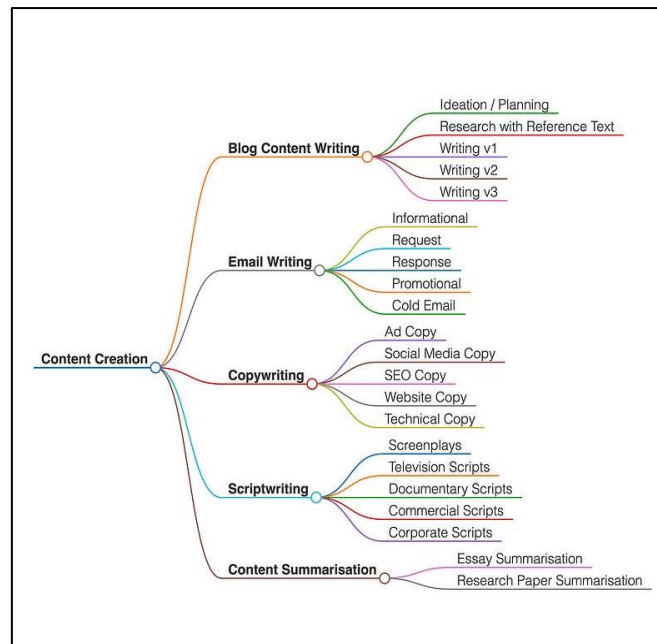
The applications of LLMs are vast and diverse, spanning across multiple industries and domains. From natural language processing and text generation to chatbots and virtual assistants, LLMs are revolutionizing the way we interact with technology and unlock new possibilities for automation and efficiency. As these powerful models continue to evolve, we can expect to see even more innovative LLM applications emerge, further transforming the landscape of artificial intelligence and its impact on our daily lives.

The usage cases for LLMs are vast and continue to expand with the release of each new model. In this article we provided you with several examples of how LLMs can be used: **sentiment analysis, question answering/chatbot, content generation, summarization, text classification**, and **cybersecurity**. This is in no way the limit to their usage, however these categories represent the main ways LLMs are being used today. As the quality of LLMs improves, new and innovative uses will be discovered.

X. CONTENT CREATION

- Large language models (LLMs) are transforming the landscape of content creation. These powerful AI models can generate human-like text, translate languages, write different types of creative content, and even debug code. As a result, LLMs are having a major impact on a wide range of industries, from marketing and journalism to education and entertainment.
- One of the most significant ways that LLMs are revolutionizing writing is by automating the content creation process. LLMs can be used to generate high-quality content quickly and efficiently, freeing up human writers to focus on more creative and strategic tasks. For example, LLMs can be used to generate blog posts, articles, social media content, and even entire books.
- LLMs are also being used to create new and innovative types of content. For example, LLMs can be used to generate personalized stories, interactive chatbots, and even video games. As LLMs continue to develop, we can expect to see even more groundbreaking applications of this technology in the years to come.
- Of course, LLMs are not without their challenges. One of the biggest challenges is ensuring that LLM-generated content is accurate and unbiased. Additionally, there are concerns about the potential for LLMs to be used to spread misinformation and propaganda.
- Despite these challenges, LLMs have the potential to revolutionize the way that we create and consume content.

Content Creation Usecase



LLM-powered chatbots handle customer inquiries accurately, voice bots offer real-time customer service, and video bots provide interactive video content, especially in healthcare, for virtual consultations. LLMs deliver real-time content responses with an average latency of under two seconds.

Impact on content creation LLMs have transformed content creation, automating tasks that were once time-consuming. McKinsey reports that AI-driven tools, including LLMs, can boost productivity by up to 40% in some industries. LLMs can generate articles, marketing copies, and technical documents with speed and precision.

In media and entertainment, they draft scripts and create news content; in healthcare, they automate patient materials and generate reports; and in e-commerce, they personalise product descriptions and recommendations.

With over 85% accuracy in tailoring content to user preferences, LLMs are revolutionising the way content is created and delivered across sectors.

Content Generation Patterns

LLMs are used by applications to generate a wide range of content. When it comes to categorization of the content, it can range from total creative content with minimal guidance to extracting specific information with a high degree of guidance. Let's look at the LLM Content Generation patterns.



Creative Text Composition

This category involves open-ended text generation, where a LLM is given minimal guidance or constraints allowing it to unleash its creativity based on the input prompt. We let the LLM to create (hallucinate) to large extent without narrow specifications on output length, format, style or content genres. This can be instrumental for tasks requiring more of open-ended contents, such as blog posts, imaginative storytelling, or creative marketing content. LLMs are trained with massive volume of data with content cutting across every possible categories-academia, web, news, research etc. They can generate content swiftly with unconventional ideas or angles. However, the content might lack human finesse or nuanced understanding in certain contexts.

Content Compilation

In this category, the LLM digests the information contained within provided context and documents, then compiles a synthesized summary reflecting the key aspects. This leverages the LLM's ability to analyze texts and extract salient points into a coherent digest form. The LLM has degrees of freedom to create the compiled content as long as the semantic meaning is retained. Example applications are legal contract summaries, extractive summaries, from academic papers or articles, or meeting/notes compilation. LLMs excel at sifting through large volumes of information swiftly, summarizing key points accurately. As compared to humans, LLMs can process and synthesize information from multiple sources more efficiently, providing concise summaries that save time and effort.

Targeted Information Retrieval

Here the LLM is focused on retrieving specific and precise information within a provided context, this category requires a high degree of guidance. It analyzes contextual documents to extract relevant information to directly address the query, generating an informative response. Examples are traditional Q&A capabilities, precise information extraction from texts (e.g. Who is the author?), classification tasks and output in the expected format (E.g. JSON).

Cross-Domain Proficiency

This category involves generation of content in diverse domains such as translation, code generation, abstractive summarization, domain specific content generation (e.g. Generating medical report patient's symptoms and lab records) etc. Here the LLM creates content that possesses semantic meaning in a different knowledge domain compared to the original context. This made possible again with vast amount of training data from multiple disciplines and LLM can adeptly navigate between domains across different subject matters, languages, or technical domains.

Interactive Generation

This is a unique category depicting Inversion of Control (IOC), where the LLM assumes the role of determining the flow of execution. LLM is provided with interfaces (e.g. Tools) to query the environment, make observations and take a set of actions. This powers LLMs with more advanced back-and-forth interactions. The model does not just react based on a single static prompt, but allows dynamic dialog where it requests clarifying information before formulating responses tailored to the evolving context. Prompting techniques such as ReACT and Tree of Thoughts(ToT) exemplify this capability.

It should be noted that the categories can overlap in various combinations rather than being mutually exclusive. For example, a conversational response could incorporate open-ended text generation reflecting parts of categories Content Text composition and Interactive Generation. This allows for a more flexible, accurate framework.

Practices for Creating Content Using Large Language Models

Whether you use ChatGPT or something else, creating content with large language models (LLMs) can be a daunting task. But with the right strategies in place, it can be easier.

Setting a clear aim and understanding the model's limitations are key steps, as are choosing a hybrid strategy and monitoring quality standards. But that's not all. You need to provide context and directions, refine output gradually, plus verify and cross-check any facts. They're all critical steps in the process.

1. Set a Clear Aim and Intention

Having an established aim from the start provides focus, applicability, and uniformity in your interactions with a large language model. You can customize the prompts, questions, or limits you give the model to get the preferred output. This customization boosts the applicability and quality of the created content. Establishing a determined goal simplifies evaluating and assessing the success of whatever you create.

2. Understand the Model's Limitations

Grasping a large language model's weakness is a vital step in content formation. Doing this allows you to make use of the language model more efficiently, reduce its handicaps, and raise the total quality and dependability of the generated content. It helps set realistic expectations,

find errors, interpret material in context, enhance prompts, apply ethical judgment, and advance the model step by step.

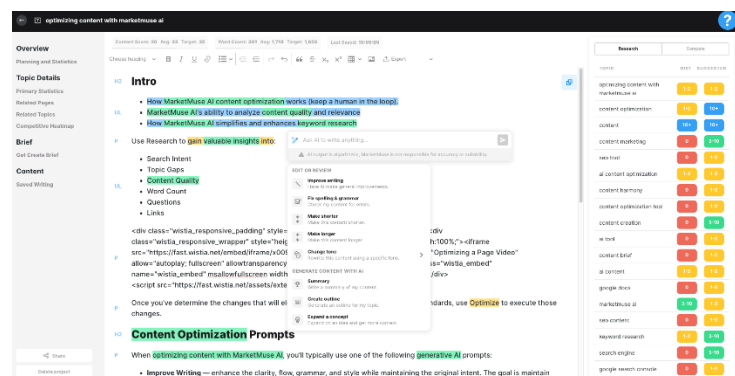
3. Choose a Hybrid Strategy

A hybrid strategy to content production combines the capabilities of large language models with human management and involvement. Time and resource efficiency are the biggest elements in using large language models. But keeping humans in the loop provides quality control, contextual understanding, and the fine-tuning of answers.

Plus, humans are better prepared to handle ethical considerations, and incorporate creativity and originality. Ultimately, humans are responsible for using the output of an LLM, so it makes sense that they have a major role.

There are a couple of ways to do this:

- Let the LLM do it's thing and spend time generating and fine-tuning your prompt. Then edit the the output to reflect your company's norms and standards. In this scenario, an editor also must be a subject matter expert, which can be a challenging position to fill. Plus, the output may be of such poor quality that you're better off starting from scratch as opposed to editing yourself out of that mess.
- Use the LLM interactively during the content creation process. Think of it as a sophisticated writer's assistant that can help create outlines, expand concepts, and summarize multiple points into one cohesive whole. In this case, the writer has an expanded role. But it's one that should be easier to fill than finding subject matter experts that are also great editors.

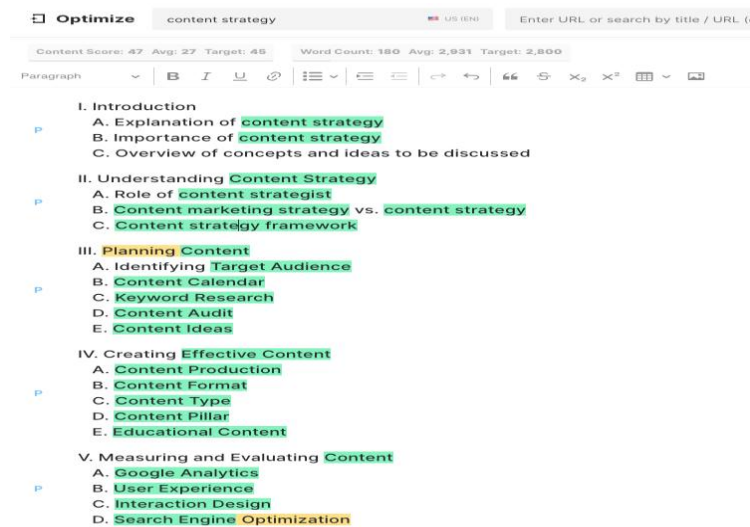


4. Monitor Quality Standards

Although large language models are trained on huge amounts of data, that doesn't guarantee that the output:

- Is always accurate and impartial.
- Gives complete coverage.
- Corresponds to the reading grade level of your crowd.
- Is of a suitable word count.

By keeping an eye on these metrics, content creators can make sure that their content is of the best quality and accuracy, while helping to find possible areas for betterment.



5. Supply Context and Directions

This step is essential in making sure that a large language model produces precise, dependable, and applicable output that aligns with your goals. It's another reason you want to begin with a clear aim and intention. By expressing your desired objectives, details, or restrictions, you can guide the model towards providing more suitable content and limit unwanted behavior or biases. It also makes it easier to satisfy those quality metrics you're tracking.

6. Gradually Refine the Output

This strategy helps improve accuracy and applicability, customize and optimize content, detect and fix mistakes, and give quality assurance and control. It also supports a continuous learning and progress attitude, allowing you to adjust to changing user requirements and preferences. This active engagement in the content creation process guarantees that the generated output meets your desired standards of accuracy and quality.

Again, this makes tracking those quality metrics a no-brainer. Plus, this approach fits well with a writer who is interacting with the LLM, iterating on its output.

7. Verify and Cross-Check

Large language models never lie, but that doesn't mean they tell the truth. Although they excel at mimicking human intelligence, they have no concept of truth or falsehood.

So verifying and checking content produced by language models is essential for accuracy, reliability, and trustworthiness. It also helps address biases, identify errors and gaps, encourage responsible content creation, and observe legal and ethical considerations. Plus, this strategy demonstrates professional integrity and builds trust with your audience.

Don't land in hot water like this lawyer who used ChatGPT to self-verify the court citations it created. When asked, ChatGPT claimed that the citations it created were real, when in fact they weren't. Always use an external source, preferably a human, to validate any facts.

Future of content creation

The impact of LLMs on content creation across industries is profound and far-reaching. By automating routine tasks, personalising user experiences, and enabling new forms of creativity, LLMs are not just tools for efficiency, but they are also catalysts for innovation.

As these models continue to evolve, their potential to transform industries will only grow, making them indispensable assets in the digital age. Whether in media, healthcare, ecommerce, legal, or education, LLMs are setting new standards in content creation.

The future of content creation is here, and it's powered by large language models.

XI. CHATBOT SYSTEM

At the most basic level, a chatbot is a computer program that simulates and processes human conversation (either written or spoken), allowing humans to interact with digital devices as if they were communicating with a real person. Chatbots can be as simple as rudimentary programs that answer a simple query with a single-line response, or as sophisticated as digital assistants that learn and evolve to deliver increasing levels of personalization as they gather and process information.

You've probably interacted with a chatbot whether you know it or not. For example, you're at your computer researching a product, and a window pops up on your screen asking if you need help. Or perhaps you're on your way to a concert and you use your smartphone to request a ride via chat. Or you might have used voice commands to order a coffee from your neighbourhood café and received a response telling you when your order will be ready and what it will cost. These are all examples of scenarios in which you could be encountering a chatbot.

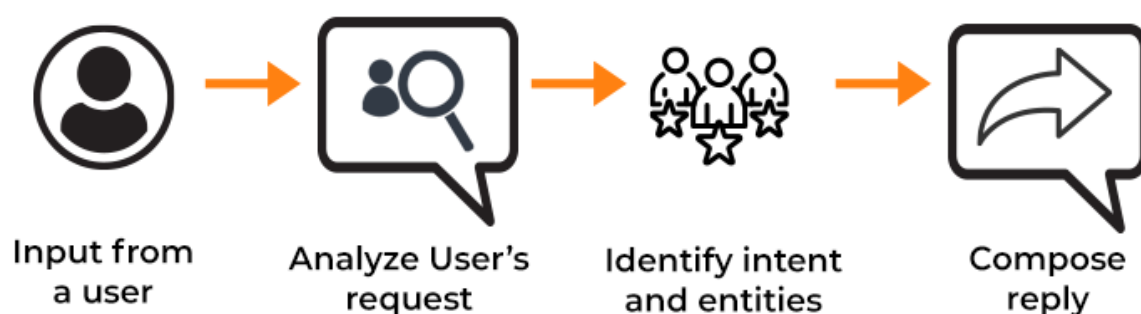
Driven by AI, automated rules, natural-language processing (NLP), and machine learning (ML), chatbots process data to deliver responses to requests of all kinds.

- **Task-oriented (declarative) chatbots** are single-purpose programs that focus on performing one function. Using rules, NLP, and very little ML, they generate automated but conversational responses to user inquiries. Interactions with these chatbots are highly specific and structured and are most applicable to support and service functions—think robust, interactive FAQs. Task-oriented chatbots can handle common questions, such as queries about hours of business or simple transactions that don't involve a variety of variables. Though they do use NLP so end users can experience them in a conversational way, their capabilities are fairly basic. These are currently the most commonly used chatbots.

- **Data-driven and predictive (conversational) chatbots** are often referred to as virtual assistants or digital assistants, and they are much more sophisticated, interactive, and personalized than task-oriented chatbots. These chatbots are contextually aware and leverage natural-language understanding (NLU), NLP, and ML to learn as they go. They apply predictive intelligence and analytics to enable personalization based on user profiles and past user behaviour. Digital assistants can learn a user's preferences over time, provide recommendations, and even anticipate needs. In addition to monitoring data and intent, they can initiate conversations. Apple's Siri and Amazon's Alexa are examples of consumer-oriented, data-driven, predictive chatbots.

Advanced digital assistants are also able to connect several single-purpose chatbots under one umbrella, pull disparate information from each of them, and then combine this information to perform a task while still maintaining context—so the chatbot doesn't become “confused.”

HOW AN AI CHATBOTS WORKS



To understand how a chatbot works, we must first consider the three core mechanisms driving the technology. The three mechanisms that require your attention are rules-based processes, AI-driven decision-making, and live agent intervention. Depending on a chatbot's mechanism, its functionality will be slightly different.

- **Rules-based processes**

Rules-based chatbot software performs pre-programmed behaviors based on “playbooks” you create on the user interface's backend module. Like a digital assistant, rules-based chatbot technology can behave in a certain way based on click activities and simple event triggers like a “yes” or a “no” input. It may also detect a specific keyword or combination of phrases (but only when there is an exact match).

For instance, you could program a rules-based chatbot to answer not only if someone chooses “black” or “white,” but also if they say “I want a black item” – the chatbot's backend module will match the term black with a preconfigured rule.

- **AI-driven decision-making**

Artificial intelligence chatbots employ AI and natural language processing (NLP) technology to recognize sentence structure, interpret the knowledge, and improve their ability to answer questions.

Instead of relying on a pre-programmed response, AI chatbots first determine what the customer or user is saying. Then, once they have figured out what the user is looking for, the chatbot provides an answer that it believes is correct based on the available data. The machine learns the “right” response over time by analysing correct and erroneous responses.

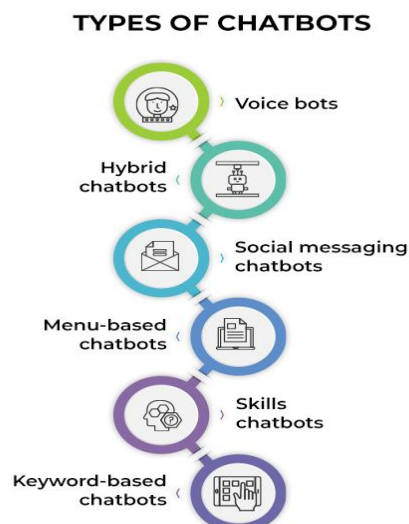
With AI-driven decision-making mechanisms, a chatbot can be extremely effective, provided they have a thorough understanding of your organization, its customers, and its context. This functionality is primarily used by large enterprises like e-commerce as well as other high-volume businesses that demand scale.

- **Live agent interaction**

Live chat is a type of chat system that sits on the webpage or in your mobile application and works as a consumer’s window to your support team and contact center. Using this mechanism, chatbots incorporate routing capabilities to assign discussions in real-time.

When a customer needs to communicate with a representative from your team, the chatbot scans agent availability and routes the discussion request accordingly. It will connect the customer with someone who can help them with their problem – i.e., an agent with the right skills and knowledge. The chatbot also alerts the agent when there is a customer query and informs the customer about agent details like their name, waiting time, etc.

As you can see, these processes are relatively understandable, given that advancements in chatbot technology today are endless and readily accessible to users and developers alike.



1. Voice bots

A voice bot is a voice-to-text and text-to-speech communication channel powered by AI and natural language understanding (NLU). AI technology aids in identifying key speech signals and determining the optimal conversational response. The text-to-speech (TTS) engine subsequently completes the interaction by converting the message into audio or voice.

These bots are programmed to complete the entire speech comprehension and response process in a human-like manner. Voice assistants or voice chatbots provide a sophisticated model of communication that one may quickly implement into various customer service tools, including interactive voice response (IVR), self-service, and online knowledge bases.

2. Hybrid chatbots

A hybrid chatbot is a harmonious blend of chatbot and live chat that combines the best of both worlds. A customer service representative will be available in live chat to answer any customer's questions, which may be too complex or nuanced for automation alone.

An AI component in a chatbot replicates conversations based on how it is programmed and the needs of the conversation. On the other hand, a hybrid chatbot will initiate an automated chat conversation and attempt to resolve the user's query as quickly and simply as possible. If it does not function as expected, a customer service representative can intervene at any moment or in the subject matter area where the chatbot cannot complete the task.

3. Social messaging chatbots

With the rise of new social media interfaces, organizations can now deploy an AI algorithm across all of their customer's preferred messaging platforms. This includes Facebook Messenger, Twitter, Instagram, as well as messaging apps like WhatsApp and WeChat. It enables a more pleasant online experience for customers and increased engagement for the company – all without adding to a contact center's workload.

4. Menu-based chatbots

The most rudimentary type of chatbot in use is one that is based on menu-driven navigation. Most of the time, these chatbots follow a fixed decision tree that is displayed to the consumer in the form of clickable buttons. These chatbots (like the automated dial pad menus on telephones that we use regularly) ask the user to make several choices and click on suitable options to get to the final solution.

Although these chatbots are adequate for addressing frequently asked questions, which account for most support requests, they may fall short in more complicated scenarios. If there are too many elements or too much expertise at play, the menu-based chatbot may not be able to help users arrive at the correct response. It is also important to note that menu-based chatbots are the slowest to deliver genuine value to the consumer, but they are simple and affordable to get started.

5. Skills chatbots

A skills chatbot is another kind of bot that can perform a specific set of tasks, once you have extended its capabilities using pre-defined skills software. For example, the chatbot may be able to provide weather information, turn off your room lights when connected to a smart home

appliance, order groceries online, etc. With access to the skill's source code, developers can construct their own skills chatbots and integrate them with other platforms.

6. Keyword-based chatbots

Keyword-based chatbots can listen to what visitors enter and answer correctly, unlike menu-based chatbots. These chatbots use customizable keywords and NLP to detect action triggers in the conversation to understand how to respond appropriately to the consumer. However, when faced with many similar inquiries, these chatbots may fall short. The chatbots could begin to falter if there are keyword repetitions across numerous associated inquiries.

That is why chatbots that combine keyword identification and menu or button-based navigation are becoming increasingly popular. If the keyword detection functionality fails or the user needs additional help finding an answer, such chatbots give users the option of directly entering commands via clickable navigation buttons. This is an effective workaround when the bot cannot detect keywords in the typed input.

7. Rules-based chatbots

A rule-based chatbot is ideal for companies that already know the types of inquiries their customers will ask. Chat flows are created by using if/then logic, and you must first establish the chatbot's language requirements. Conditions for evaluating words, word structure, synonyms, and more are the essential tenets of its functionality. Customers will receive prompt assistance if an inbound inquiry falls within these parameters.

It is important to note that it is the developer's responsibility to cover every permutation and combination of a query as much as possible – or else, the chatbot will not be able to understand the consumer or respond to them.

8. AI-powered contextual chatbots

Contextual chatbots can grasp the context of a chat and determine the correct meaning of the user inquiry. It can also recall prior interactions and use that information to maintain relevance while interacting with repeat customers. Contextual bots can guarantee that repeat users have a consistent experience. Furthermore, it may retain information on user intent gathered across numerous platforms and channels, ensuring that the conversation's context matches the needs of the consumer at every touchpoint.

Contextual chatbots are connected to the centralized database of a site or app, typically a customer relationship management (CRM) system or a customer data platform (CDP). This enables them to retrieve critical information about an individual with whom they are chatting, such as the individual's name, location, or purchase history.

9. Support chatbots

Support chatbots are conversational systems designed solely to provide customer support and post-purchase services. Unlike bots on social media or websites, they do not share offers, promos, or other customer engagement materials. This type of chatbot is typically found on self-service portals and online documentation, where users might come to receive support and help. Support chatbots are widely used for internal purposes, including answering HR queries, raising IT tickets, submitting employee documents, etc.

10. Transactional bots

Transactional chatbots can help organizations strengthen their sales and marketing efforts, whether for appointment scheduling, lead generation, or payment collection. Users can conduct transactions directly while conversing with the chatbot without human intervention.

Its biggest benefit is the ability to enable transactions and drive business 24/7/365. As a result, a transactional chatbot differs from other types of bots, like informational or support bots. Its focus is to complete a transaction and streamline the user experience by offering a quick and easy channel for a single purpose. It is designed to handle a small number of specialized tasks.

11. No code or low code chatbots

Chatbots have traditionally been designed and developed using code to create decision trees and AI and machine learning (ML) algorithms powering the technology. Each programming language has a web API that one can use to build chatbots. In addition to PHP and Node.js, many other libraries that enable Python or Java are used in most typical deployments.

However, recent advancements allow organizations to use chatbots that require little or no coding. This allows for speedier application delivery and faster value generation since a graphical user interface (GUI) is available to build and configure the bot. No-code deployments are suitable for information-collecting chatbots and those that encourage human interaction. In contrast, low-code chatbots are ideal for organizations that need to add unique features while reducing development efforts.

The future of chatbots

Where is the evolution of chatbots headed? Chatbots, like other AI tools, will be used to further enhance human capabilities and free humans to be more creative and innovative, spending more of their time on strategic rather than tactical activities.

In the near future, when AI is combined with the development of 5G technology, businesses, employees, and consumers are likely to enjoy enhanced chatbot features such as faster recommendations and predictions, and easy access to high-definition video conferencing from within a conversation. These and other possibilities are in the investigative stages and will evolve quickly as internet connectivity, AI, NLP, and ML advance. Eventually, every person can have a fully functional personal assistant right in their pocket, making our world a more efficient and connected place to live and work.

References:

Topic	Source
How transformer is suitable for NLP	https://datasciencedojo.com/blog/transformer-models/
Encoder only LLMs : BERT,	https://medium.com/@pickleprat/encoder-only-architecture-bert-4b27f9c76860
Decoder only LLMs : GPT, Llama,	<ol style="list-style-type: none"> https://ai.stackexchange.com/questions/40179/how-does-the-decoder-only-transformer-architecture-work https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse https://medium.com/@lmpo/decoder-only-transformer-models-a-comprehensive-overview-c54dc3286d71
MoE based Architecture	<ol style="list-style-type: none"> https://medium.com/@mne/explaining-the-mixture-of-experts-moe-architecture-in-simple-terms-85de9d19ea73#:~:text=The%20Mixture%20of%20Experts%20(MoE)%20101&text=MoEs%2C%20unlike%20traditional%20dense%20models,managing%20a%20larger%20model%20size. https://www.datacamp.com/blog/mixture-of-experts-moe https://www.exactcorp.com/blog/deep-learning/why-new-llms-use-moe-mixture-of-experts-architecture https://zilliz.com/learn/what-is-mixture-of-experts
Significance of decoder only LLMs over encoder only LLMs	https://chatgpt.com/
Translation in NLP	<ol style="list-style-type: none"> https://www.scaler.com/topics/nlp/machine-translation-in-nlp/ Text Book: Natural Language Processing with Transformers
Summarization in NLP	<ol style="list-style-type: none"> https://addepto.com/blog/text-summarization-using-nlp-techniques-and-use-cases/ https://medium.com/analytics-vidhya/text-summarization-using-nlp-3e85ad0c6349 Text Book: Natural Language Processing with Transformers
Named Entity Generation (NER)	<ol style="list-style-type: none"> https://medium.com/analytics-vidhya/journey-of-named-entity-recognition-282587e126b7 Text Book: Natural Language Processing with Transformers
Applications	https://medium.com/ubiops-tech/top-7-current-llm-applications-and-use-cases-567f3c9f07e3
Content Creation	<ol style="list-style-type: none"> https://medium.com/@gopikwork/unlocking-the-potential-of-llms-content-generation-model-invocation-and-training-patterns-c84c23e6aeb0 https://blog.marketmuse.com/content-creation-best-practices-large-language-models
Chatbot System	<ol style="list-style-type: none"> https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-chatbot/ https://www.oracle.com/in/chatbots/what-is-a-chatbot/#:~:text=At%20the%20most%20basic%20level,communicating%20with%20a%20real%20person.