**Backprop Understanding in Simple Words**
Most deep learning libraries provide a fast and flexible framework for building deep learning projects by easing the computation of the partial derivatives (also called gradients) that drive learning through backpropagation.

When training a model, we use a loss function to measure how well the model's predictions match the target values (the true labels or expected outcomes). The model prediction is what the model outputs based on the current weights, and the target is the actual expected result.

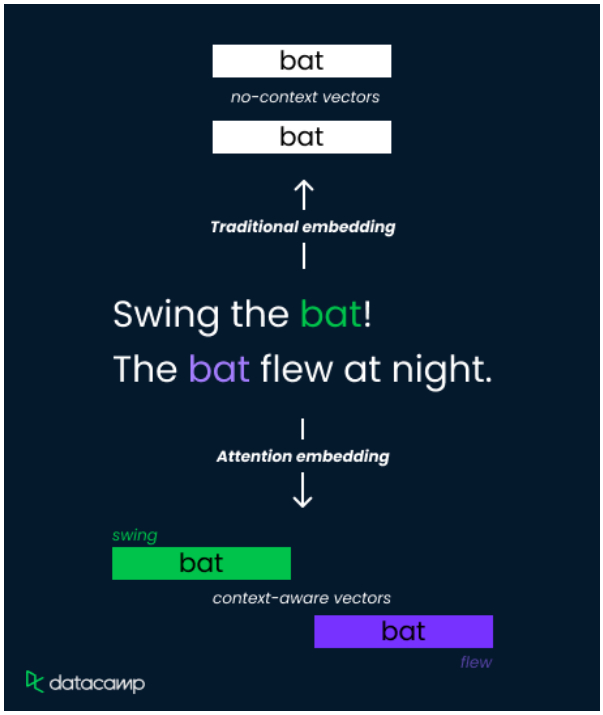
The loss function computes the error between the model's predictions and the target values. This error is a measure of how far the predictions are from the targets. The goal of training is to minimize this error, making the predictions as close as possible to the targets.

The gradients of the loss function are computed with respect to the model's parameters (weights). These gradients indicate the direction and magnitude of the changes needed to adjust the weights to minimize the loss function. By updating the weights in the direction that reduces the loss, the model improves its predictions over time.

**LLM, RAG, interview questions**

- The **attention mechanism** was introduced in 2017 in the paper Attention Is All You Need. Unlike traditional methods that treat words in **isolation**, attention assigns weights to each word based on its **relevance to the current task**. This enables the model to capture long-range dependencies, **analyze both local and global contexts** simultaneously, and resolve ambiguities by attending to informative parts of the sentence.
- The basics of language processing: Language models process language by trying to understand grammatical structure (syntax) and meaning (semantics).
    - **Parsing**: This technique analyzes the sentence structure, assigning **parts of speech** (noun, verb, adjective, etc.) to each word and identifying grammatical relationships.
    - **Tokenization**: The model **splits sentences into individual words** (tokens), creating the building blocks for performing semantic analysis
    - **Stemming**: This step reduces words to their **root form** (for example, "walking" becomes "walk"). This ensures the model treats similar words consistently.
    - **Entity recognition** and relationship extraction: These techniques work together to identify and **categorize specific entities** (like people or places) within the text and uncover their relationships.
    - Word **embeddings**: Finally, the model **creates a numerical representation for each word (a vector)**, capturing its meaning and connections to other words. This allows the model to process the text and perform tasks like translation or summarization.
- The limitations of traditional models

    - **Limited context:** Traditional models often represented text as a set of individual tokens, **failing to capture the broader context of a sentence.**
    - **Short context**: The window of context these models considered during processing was often limited. This meant they **couldn't capture long-range dependencies, where words far apart in a sentence influence each other's meaning.**
    - **Word disambiguation** issues: Traditional models struggled to **disambiguate words with multiple meanings** based solely on the surrounding words




- Traditional word embedding techniques, such as Word2Vec and GloVe, represent words as fixed-dimensional vectors in a semantic space based on co-occurrence statistics in a large corpus of text.
- While these embeddings capture some semantic relationships between words, they lack context sensitivity. This means the same word will have the same embedding regardless of its context within a sentence or document.
- Applications
    - Machine translation
    - Text summarisation
    - Question answering
    - Sentiment analysis
    - Content generation

**LLMs Questions**

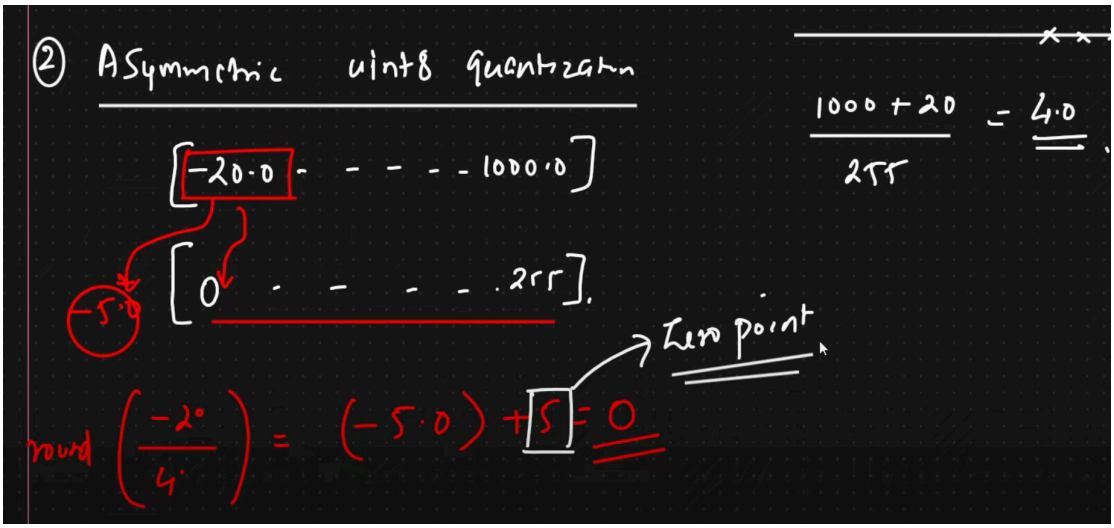
- What is the Transformer architecture, and how is it used in LLMs?
    - It relies on self-attention mechanisms to **process input data in parallel**, making it **highly scalable** and capable of capturing long-range dependencies.
- Explain the concept of "context window" in LLMs and its significance.
    - The context window in LLMs refers to the **range of text (in terms of tokens or**

**words) that the model can consider at once** when generating or understanding language. The significance of the context window lies in its impact on the model's ability to generate logical and relevant responses.
    - **A larger context window allows the model to consider more context,** leading to better understanding and text generation, especially in complex or lengthy conversations.
    -

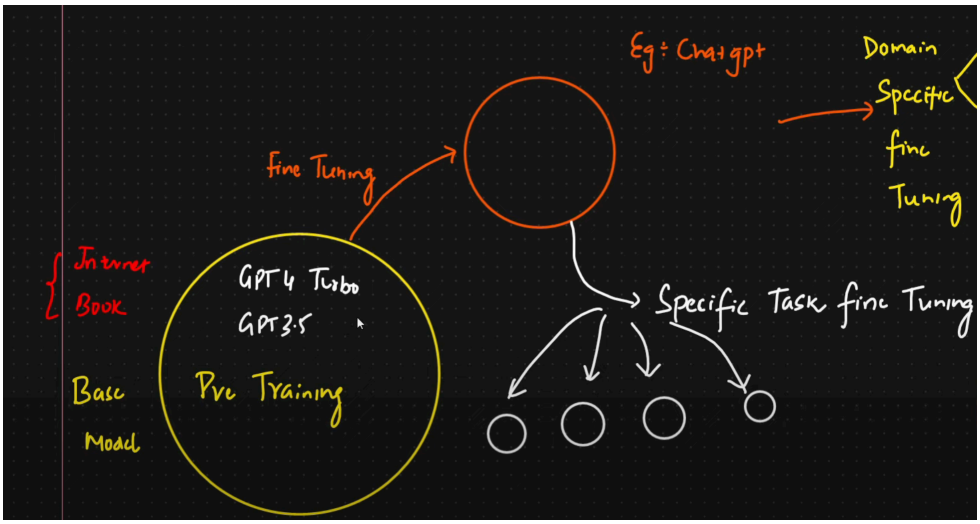
| Model | Context WIndow Size |
|---|---|
| GPT 3 | 2049 |
| GPT 4o | 128K |
| Llama 3 | 32K |
| Gemini | 128K |

    -
- What are some common pre-training objectives for LLMs, and how do they work?
    - In **MLM(Masked Language Modelling)**, random words in a sentence are **masked**, and the model is **trained to predict the masked words** based on the surrounding context. This helps the model understand the bidirectional context.
    - **Autoregressive language modeling** involves **predicting the next word in a sequence** and training the model to **generate text one token at a time**. Both objectives enable the model to learn language patterns and semantics from large corpora, providing a solid foundation for fine-tuning specific tasks.
- What are some common challenges associated with using LLMs?
    - Computational resources: LLMs require **significant computational power** and memory, making training and deployment resource-intensive.
    - Bias and fairness: LLMs can **inadvertently learn and propagate biases present in the training data**, leading to unfair or biased outputs.
    - Interpretability: Understanding and explaining the decisions made by LLMs can be difficult due to their complex and **opaque nature.**
    - Data privacy: Using large datasets for training can raise concerns about data privacy and security.
    - Cost: The development, training, and deployment of LLMs can be **expensive**, limiting their accessibility for smaller organizations.
- How do LLMs handle out-of-vocabulary (OOV) words or tokens?
    - LLMs handle out-of-vocabulary (OOV) words or tokens using techniques like **subword tokenization** (e.g., Byte Pair Encoding or BPE, and WordPiece). These techniques **break down unknown words into smaller, known subword units** that the model can process.
    - This approach ensures that even if a word is not seen during training, the model can still understand and **generate text based on its constituent parts**, improving flexibility and robustness.
- What are embedding layers, and why are they important in LLMs?

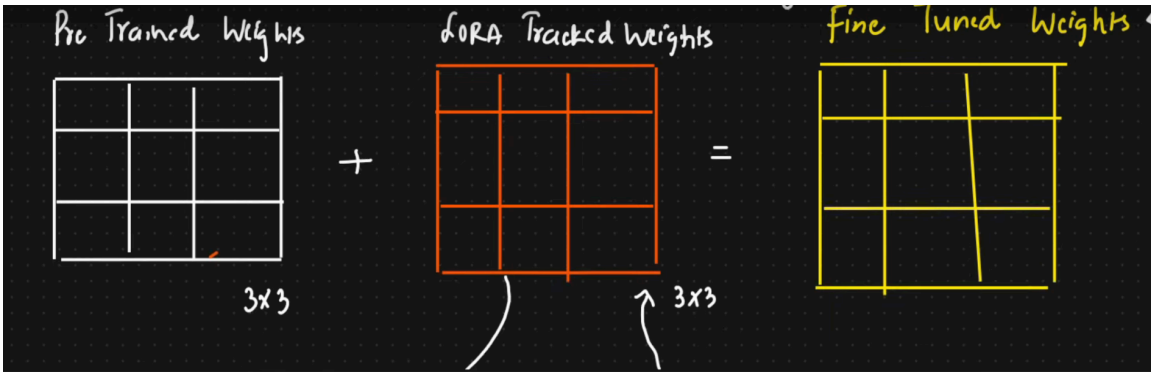
    - Embedding layers are a significant component in LLMs used to **convert categorical data, such as words, into dense vector representations**.These embeddings capture semantic relationships between words by representing them in a continuous vector space where similar words exhibit stronger proximity.

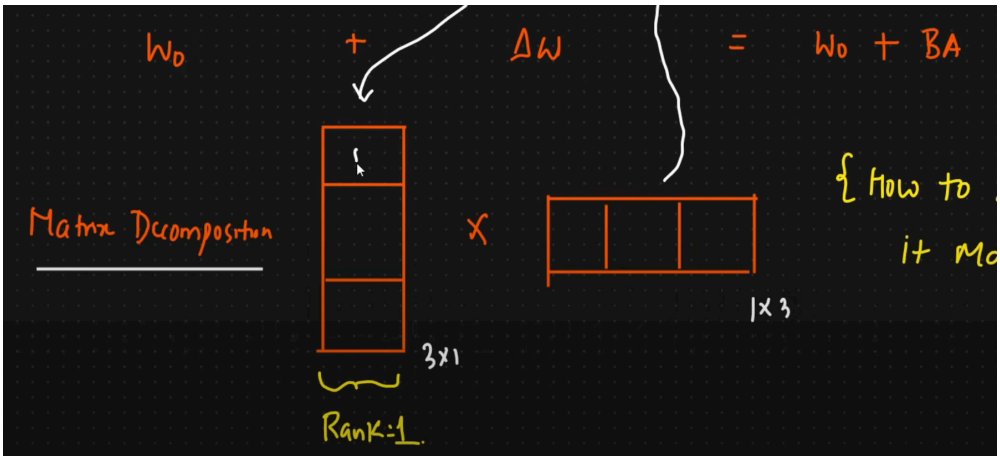
- Quantization
    - Conversion from higher memory format to a lower memory format
    - Weights are in the form of a matrix with every value store in 32bits(FP32)(Full Precision) can be converted to int 8(8 bits)
    - Limited RAM in smaller machines or mobile phones and edge devices
- How to perform quantization(calibration)
    - Symmetric quantization
        - Batch normalisation is one such technique in DL where between Forward prop and back prop, you center the weights around 0.
        - Stored in memory as sign, exponent and mantissa
        - Min max scaling is basically used to get scaling factor with (xmax-xmin)/(qmax - qmin). Higher to lower is round(higher/scaling factor)
    - Assymetric Quantization
        - 

- Post training quantization(PTQ)
    - Have a pretrained model. Apply calibration. Get quantized model
    - Loss of data and accuracy
- Quantization aware training(Q AT)
    - Have a pretrained model. Apply Calibration. Perform finetuning with new training data. Get quantized model

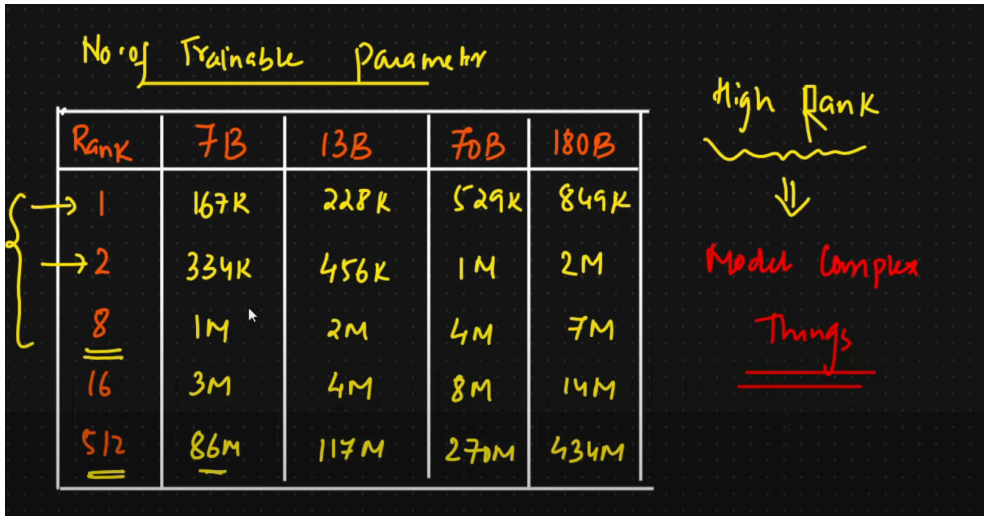
- Low rank adaptation of large language models(LoRA)
    - Used in finetuning of llm
    - 

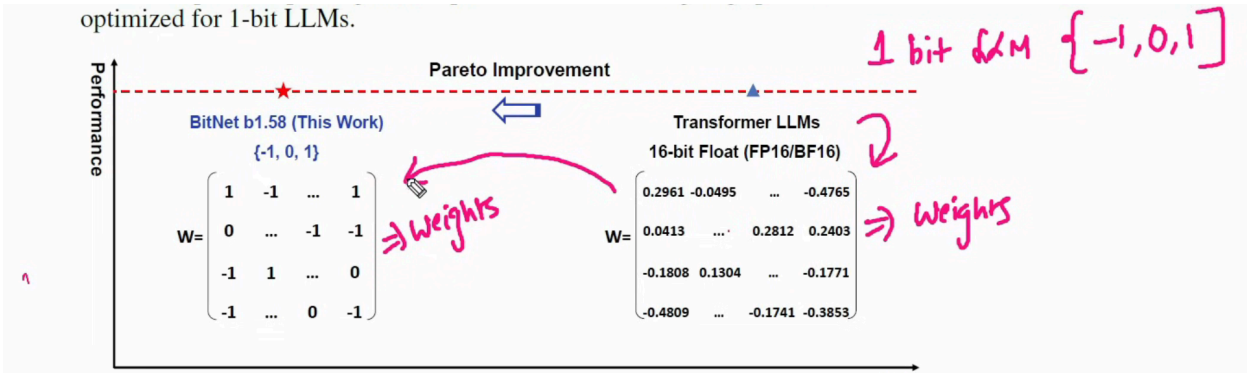
    - Full parameter finetuning:
        - Update all weights
        - Hardware and resource constraint
        - Downstream task like model monitoring, inference becomes difficult
        - To overcome we use LORA and QLORA
    - Specific task finetuning like Q&A, document, summarisation

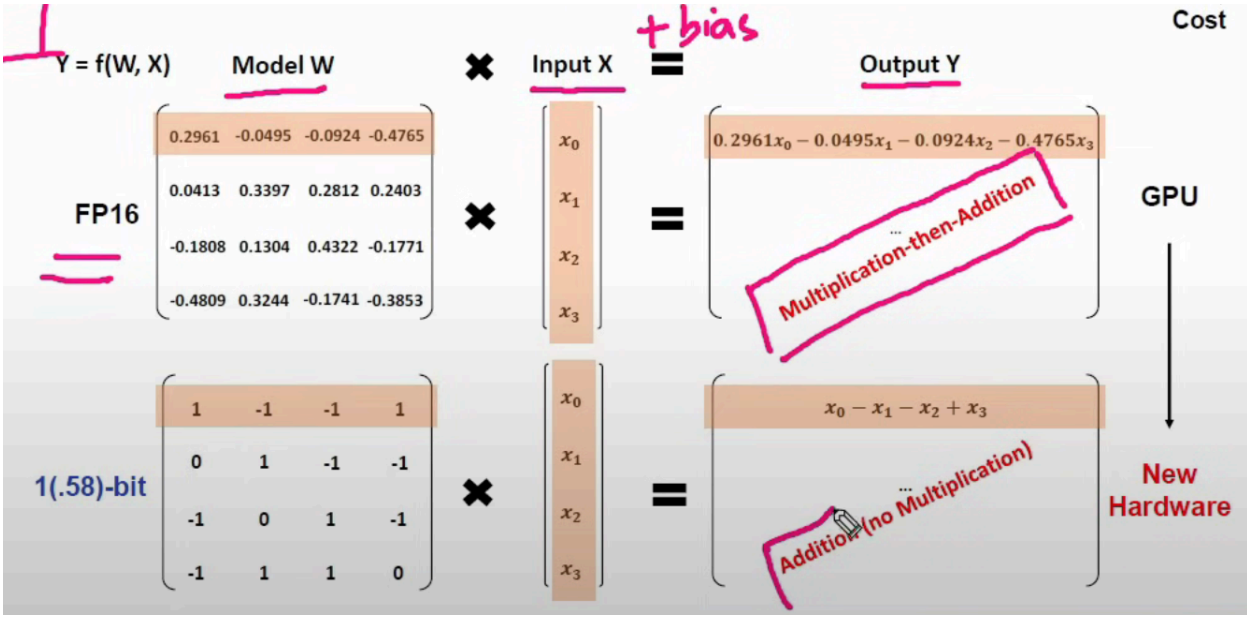
- LoRA
  - Instead of updating all the weights, it will track the changes of the new weights based on finetuning.
  - 

  - 

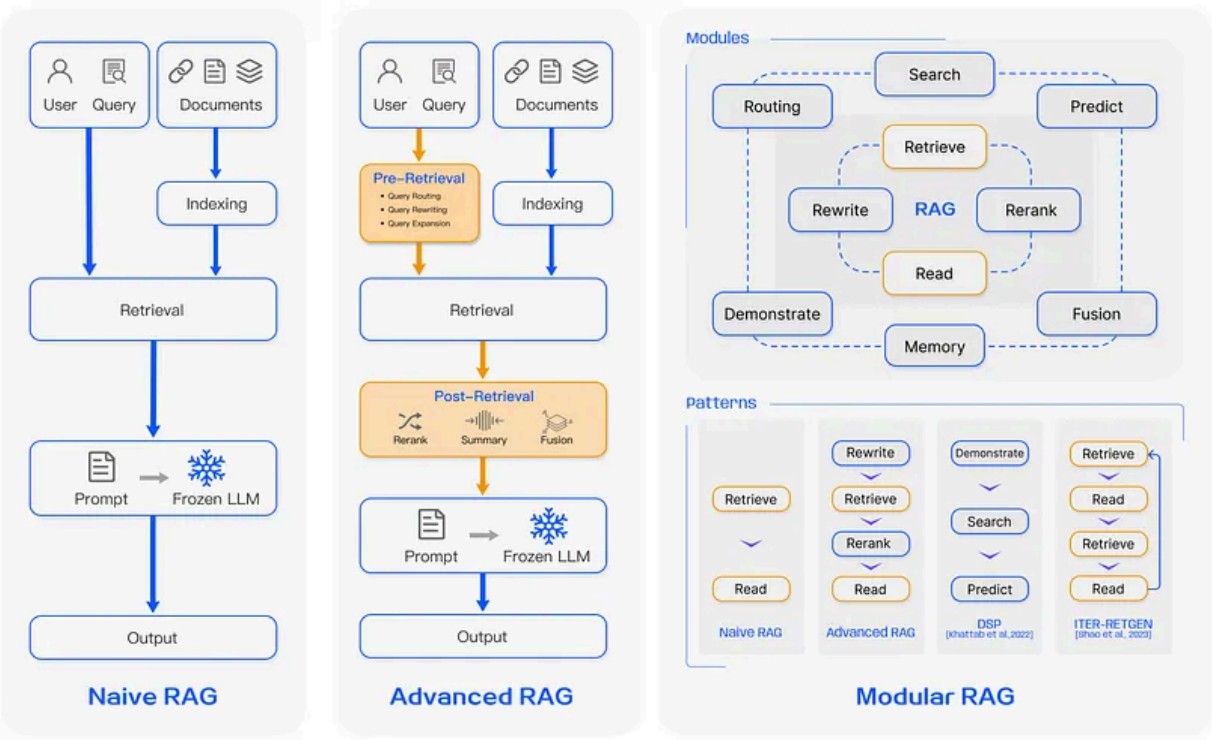
  - Based on rank of a matrix. Rank of a Matrix is defined as the dimension of the vector space formed by its columns. Rank of a Matrix is a very important concept in the field of Linear Algebra, as it helps us to know if we can find a solution to the system of equations or not. Rank of a matrix also helps us know the dimensionality of its vector space.
  - Solves the resource constraint

- 

    - QLORA(Quantized lora)
        - Weights are just quantized
- 1 bit LLMs
    - 

    - 

- RAG




### Pre-Retrieval Advantages:

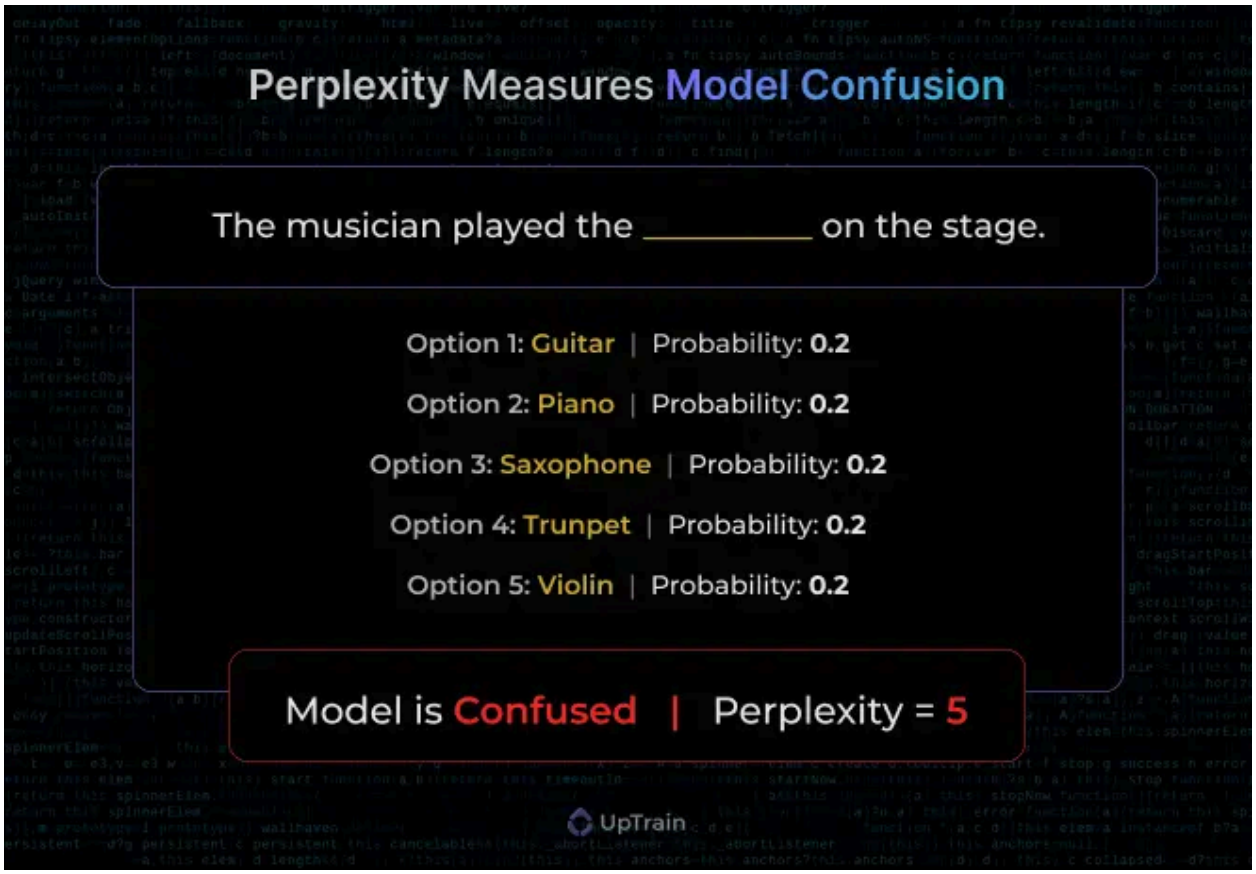
- **Optimized Index Structure**: It optimizes chunk sizes and can add graph structures to capture relationships between entities, ensuring relevant context is captured.
- **Metadata Utilization**: Advanced RAG adds relevant metadata (dates, chapters, subsections, purposes) to improve data filtering and enhance retrieval.
- **Chunk Optimization**: It optimizes chunk sizes based on document type, ensuring chunks are neither too large nor too small for better embedding and context extraction.
- **Sliding Window Chunking**: Utilizes overlapping chunks to preserve context and prevent information loss between chunks.
- **Abstract Embedding**: Prioritizes document abstracts or summaries in Top-K retrieval, providing a comprehensive understanding of the entire document.

### Retrieval Advantages:

- **Domain Knowledge Fine-Tuning**: Embedding models are fine-tuned on domain-specific datasets, capturing more accurate domain-specific semantics.
- **Optimized Similarity Metrics**: Advanced RAG allows choosing the most suitable similarity metric (Cosine Similarity, Euclidean Distance, etc.), which improves the relevance of retrieved chunks.

### Post-Retrieval Advantages:

- **Reranking**: Reranks retrieved information to prioritize the most relevant chunks, reducing performance declines caused by irrelevant or noisy context.
- **Prompt Compression**: Compresses noisy or irrelevant retrieved information, using techniques like mutual information or perplexity to reduce context length before presenting it to the LLM.
- **Summarization**: Uses summarization techniques to condense retrieved chunks, ensuring that only the most relevant information is passed to the LLM.

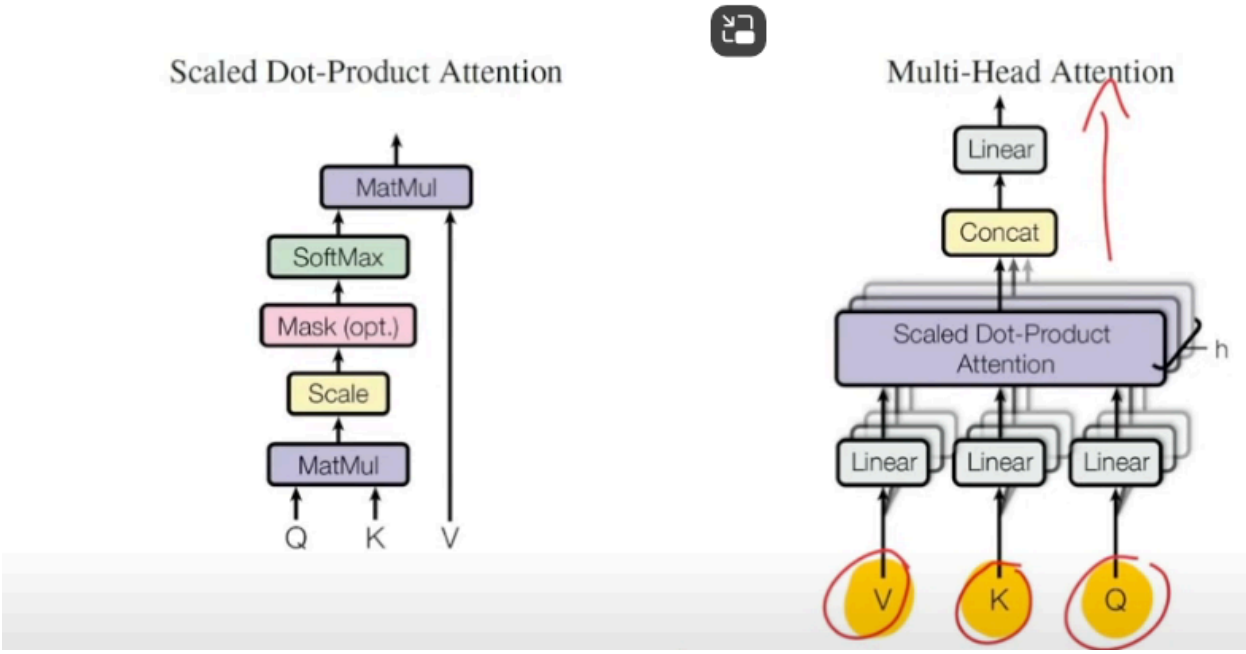
- Perplexity: measures the confidence model has in its (next word) predictions. The concept of perplexity measures how confused the model is in predicting the next word in a sequence. Lower perplexity indicates that the model is more certain about its predictions.
- 

- Reranking: Reranking is the process of **refining the initial ranking** of documents retrieved by a retrieval system. In the context of Retrieval-Augmented Generation (RAG), reranking plays a crucial role in **improving the relevance and quality of the retrieved documents** that are used to generate the final output.
    - Cross-Encoders: Cross-encoders are a popular choice for reranking in RAG. They take the **concatenated query and document as input and output a relevance score**. Examples include BERT-based models fine-tuned for passage ranking tasks. Cross-encoders can capture the interaction between the query and document effectively but are computationally expensive.

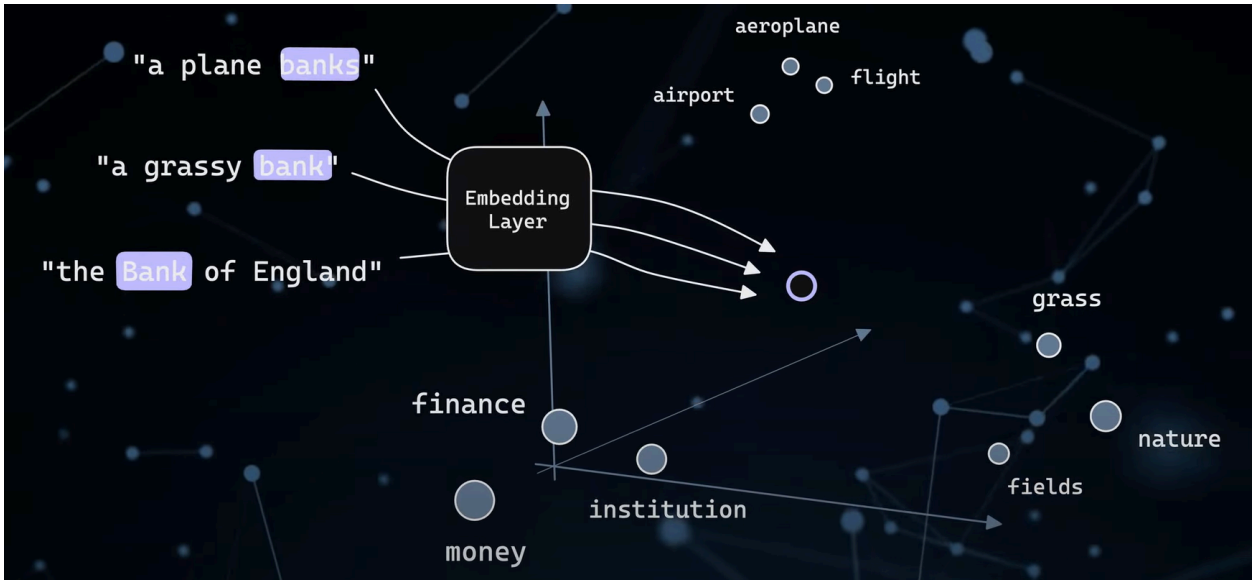
    - Bi-Encoders: Bi-encoders, also known as dual encoders, **use separate encoders** for the query and document. They **generate embeddings** for the **query and document** independently and then **compute the similarity** between them. Bi-encoders are more efficient than cross-encoders but may not capture the query-document interaction as effectively.
    - Lightweight Models: Lightweight rerankers, such as distilled models or small transformer variants, aim to strike a balance between effectiveness and efficiency. They are faster and have a smaller footprint compared to large cross-encoders, making them suitable for real-time applications.

- Prompt Engineering: Prompt engineering is a technique used to **effectively communicate** with large language models (LLM) like GPT-3 or GPT-4 and **get the desired output.**
    - Zero-Shot Learning: This involves giving the AI a task **without any prior examples**. You describe what you want in detail, assuming the AI has no prior knowledge of the task.
    - One-Shot Learning: You **provide one example** along with your prompt. This helps the AI understand the context or format you're expecting.
    - Few-Shot Learning: **This involves providing a few examples** (usually 2–5) to help the AI understand the pattern or style of the response you're looking for.
    - Chain-of-Thought Prompting: Here, you **ask the AI to detail its thought process step-by-step.** This is particularly useful for complex reasoning tasks.
    - Iterative Prompting: This is a process where you **refine your prompt based on the outputs you get**, slowly guiding the AI to the desired answer or style of answer.
    - Hybrid Prompting: **Combining different methods**, like few-shot with chain-of-thought, to get more precise or creative outputs.
    - Prompt Chaining: **Breaking down a complex task into smaller prompts** and then chaining the outputs together to form a final response.
- Cohere ranking: Rerank 3, purpose built to enhance enterprise search and Retrieval Augmented Generation (RAG) systems. Our model is compatible with any database or search index and can also be plugged into any legacy application with native search capabilities. With a single line of code, Rerank 3 can boost search performance or reduce the cost of running RAG applications with negligible impact to latency.
- Hybrid RAG:  Hybrid search combines both keyword-based methods (BM25) and vector (embedding) search techniques. Hybrid search has a specific parameter, Alpha to balance the weightage between keyword (BM25) and vector search in retrieving the right context for your RAG application. (alpha=0.0 - keyword search (BM25) and alpha=1.0 - vector search)
    - Better to tune alpha depending on number of documents, and also depends on the kind of queries the user can have.

    - BM25(Best Matching 25) scores documents based on how well they match the terms in a user's search query. The algorithm considers both term frequency (TF) and inverse document frequency (IDF), but it adds two key improvements: it normalizes term frequency to prevent the overemphasis of frequent terms, and it introduces parameters to adjust the influence of term frequency and document length on the final score.
- Types of queries
    - Web Search Queries
        - Transfer capabilities of LLaMA language model to non-English languages
    - Concept Seeking Queries
        - What is the dual-encoder architecture used in recent works on dense retrievers?
    - Fact Seeking Queries
        - What is the total number of propositions the English Wikipedia dump is segmented into in FACTOID WIKI?
    - Keyword Queries
        - GTR retriever recall rate
    - Queries With Misspellings
        - What is the advntage of prposition retrieval over sentnce or passage retrieval?
    - Exact Sub-string Searches
        - first kwords for the GTR retriever. Finer-grained
- Self-querying: Given any natural language query, the retriever uses a query-constructing LLM chain to write a structured query and then applies that structured query to its underlying VectorStore.This allows the retriever to not only use the user-input query for semantic similarity comparison with the contents of stored documents but to also extract filters from the user query on the metadata of stored documents and to execute those filters.
- Attention is All you need:



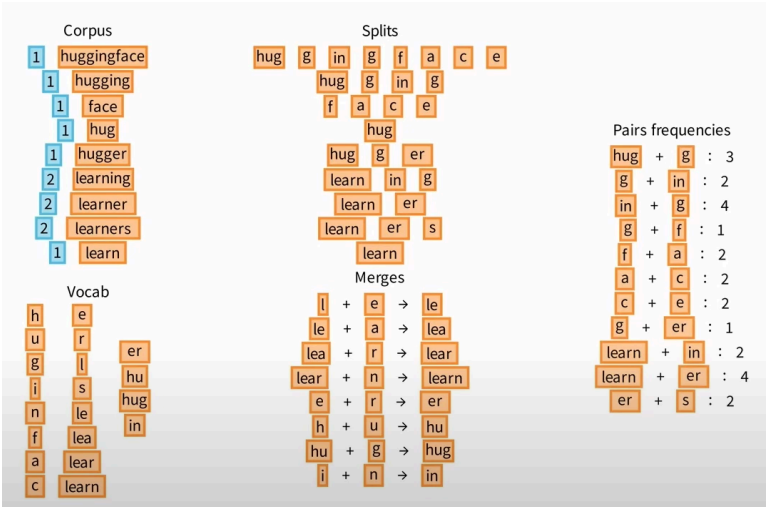

- An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed aas a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
- RNNs were not able to look at long-range dependencies and were linear so computationally inefficient.
- Encoder Decoder architecture. Encoder gets embedding and positional vector of inputs and decoder gets the targets shifted to the right so that it can learn to predict the next word based on the previous ones
- Words have 3 vectorized forms, keys, values and queries.
  - As a “query” that asks the other tokens how much attention the query token should pay to the other tokens (“keys”).
  - As a “key” so that when a “query” asks for how much attention it should pay to the input token, it can respond.
  - As a “value” that determines what content of the token should contribute to.
  - So dot product of Query and Key tells which part of the sentence holds the most relevant context and then the Value vector contains the content itself
- Multiheaded attention part ensures parellisation. So to capture as many relationships between data like grammar, subject predicate, tense, subject relation, etc. This allows for much better contextual understanding.




Based on the context around the word, the embedding layer will move it closer to flight/nature/finance in the embedding space

- Reinforcement Learning from Human Feedback (RLHF)
    - Reinforcement learning from Human Feedback (also referenced as RL from human preferences) is a challenging concept because it involves a multiple-model training process and different stages of deployment.
    - As a starting point RLHF use a language model that has already been pretrained with the classical pretraining objectives. Core to starting the RLHF process is having a model that responds well to diverse instructions.
    - OpenAI used a smaller version of GPT-3 for its first popular RLHF model, InstructGPT.
    - Generating a reward model (RM, also referred to as a preference model) calibrated with human preferences is where the relatively new research in RLHF begins.
    - Human annotators are used to rank the generated text outputs from the LM.
- Supervised Fine-Tuning (SFT)
    - Supervised Fine-Tuning is a common approach to adapt a pre-trained model to a specific task. It involves training the model on a labeled dataset, where the model learns to predict the correct label for each input.
    - This method accepts a column in your training dataset CSV that contains system instructions, questions, and answers, which form the prompt structure.
- Hallucinations
    - Instances when the model generates incorrect or nonsensical information that sounds plausible but is factually inaccurate or made up.
    - Caused by Flawed Training Data, Knowledge Gaps, Technical Limitations
    - Leading to Spreading Misinformation, Reduced Trust, Legal and Ethical Concerns
    - Prevention
        - Improving Training Data Quality, Accurate and relevant data, Remove

biases and inaccuracies
        - Human Oversight in AI Development, Human evaluators, Regular audits, User feedback
        - Future work:
            - Uncertainty estimation: Enabling LLMs to recognize when they are uncertain or lack sufficient information
            - Contrastive learning: Training LLMs to distinguish between correct and incorrect information
- Tokenizers in LLMs
    - Tokenizers serve as the foundational gatekeepers to the world of LLMs, **transforming raw text into a structured format** that these models can digest and interpret. They perform the critical task of **breaking down natural language text into manageable pieces**, known as **tokens**, which can be processed and understood by these models.
    - Tokenization is the process of **converting text into a sequence of tokens**. A token can be **as small as a character** or **as large as a word or even a sentence fragment**.
        - Word-Based Tokenization: This is one of the simplest forms, where the **text is split into tokens based on spaces and punctuation**. While straightforward, **it can struggle with languages that don't use spaces** or with complex word forms in agglutinative languages. It struggles with having a **large vocabulary (fast, faster, fastest)**, **out of vocabulary words** like mispellings or new words are a problem as well.
            - Pro: Intuitive
            - Con: Large vocab, complications with OOV and mispellings
        - Character based tokenization:
            - Pro: Small vocab, no pov
            - Con: loss of context within words, much longer sequences for input
        - Subword Tokenization: Techniques like Byte Pair Encoding (BPE), WordPiece, and SentencePiece fall under this category. These methods **break down words into smaller, more frequent subwords or symbols.** This approach **helps in handling unknown words, reducing vocabulary size, and improving model efficiency.**
            - Byte-Pair Encoding (BPE): Originally a data compression algorithm, BPE is used in tokenization to iteratively merge the most frequent pairs of characters or character sequences.
            - Widely used in various NLP tasks such as machine translation, text classification, and text generation.
                - Get a corpus of text and preprocess it
                - Create a counter to count the number of occurrence of each word
                - There is an initial vocabulary that is expanded based on the individual characters and their frequency

    - Caluclate frequencies for the pairs of characters
    - Choose the ones that appear the most frequently and create merge rules for that and add to vocabulary
    - Also account for the merge rules in the splits of the original corpus
    - By the end, the words are split into far fewer tokens
    - 

    - Pro: “Smart” vocab from frequently occurring subwords, more robust to novel words
- WordPiece:
    - Like BPE, WordPiece learns merge rules. The main difference is the way the pair to be merged is selected. Instead of selecting the most frequent pair, WordPiece computes a score for each pair, using the following formula:
    - score=(freq_of_pair)/(freq_of_first_element×freq_of_second_element)
    - The model then checks: "If I use these units (un + happiness), am I better at understanding and predicting the whole word 'unhappiness'?"
    - If the answer is yes, then the model thinks, "Great! This combination increases my likelihood of getting the right word!"
- SentencePiece:
    - Language-independent subword tokenizer
    - SentencePiece implements two subword segmentation algorithms - byte-pair-encoding (BPE) and unigram language model. These algorithms allow the tokenizer to break down words into smaller units (subwords) to reduce the vocabulary size and handle out-of-vocabulary words effectively.
    - SentencePiece treats the input text as a sequence of Unicode characters, including whitespace, which is escaped with a meta symbol (e.g., "_"). This allows for reversible encoding and decoding without losing information, making the process

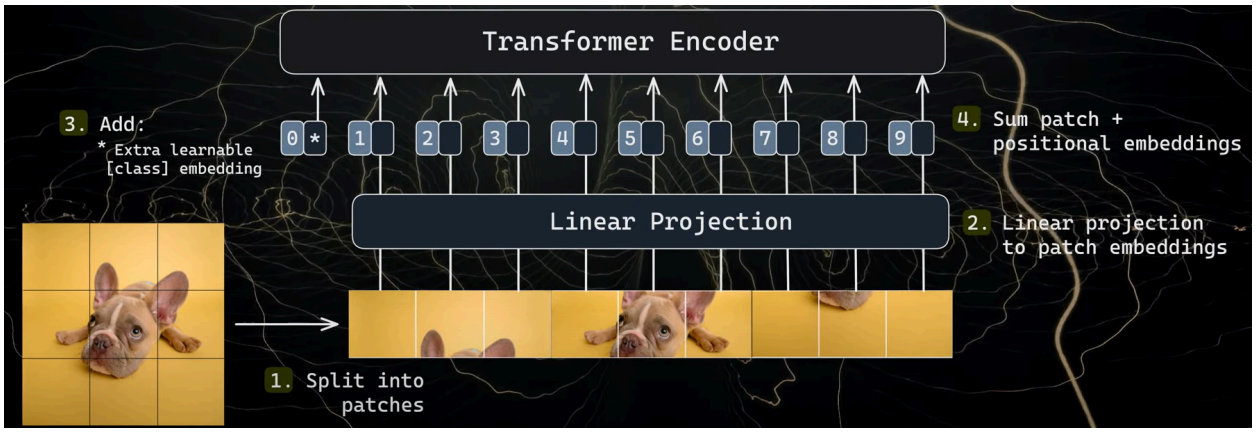
language-agnostic.

- The word "unhappiness" might be split into ["un", "happiness"], but there’s no direct information about whether "un" was a standalone unit or a prefix, making it harder to ensure proper reassembly.
- Another example: "he" and "art" might be tokenized as ["he", "art"], but during reconstruction, you might mistakenly combine these into "heart."
- For example, "I am happy" would be tokenized as ["_I", "_am", "_happy"] (where "_" represents a space).
- Because SentencePiece directly captures spaces as part of the tokenization process, it is much easier to reverse the tokenization and reconstruct the original sentence with the correct spacing.

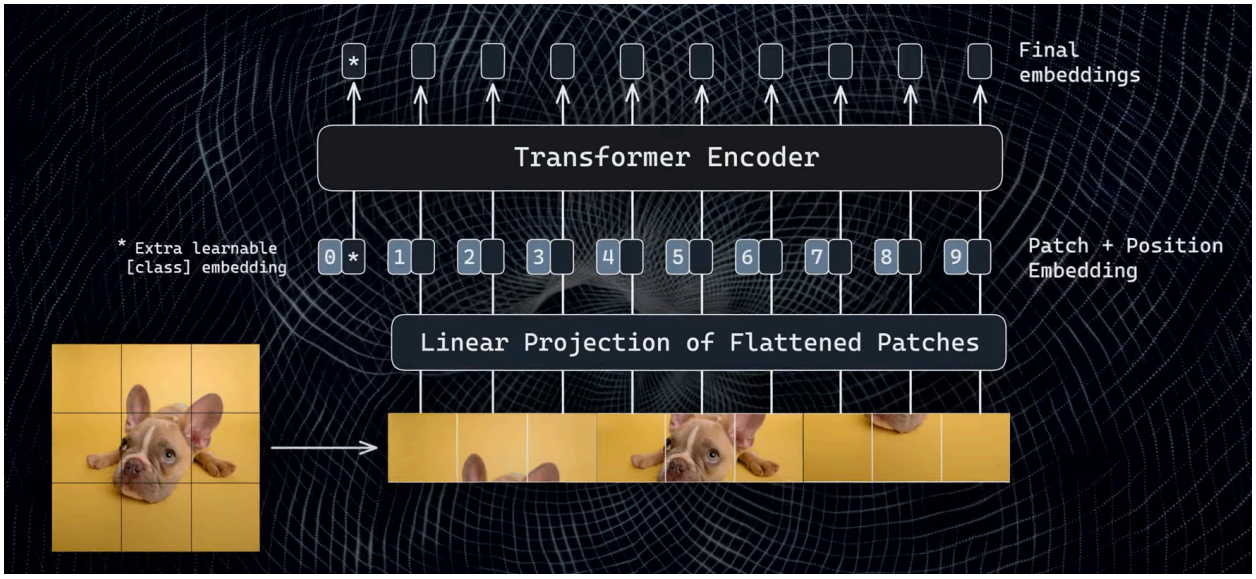
- Choosing the right tokenizer

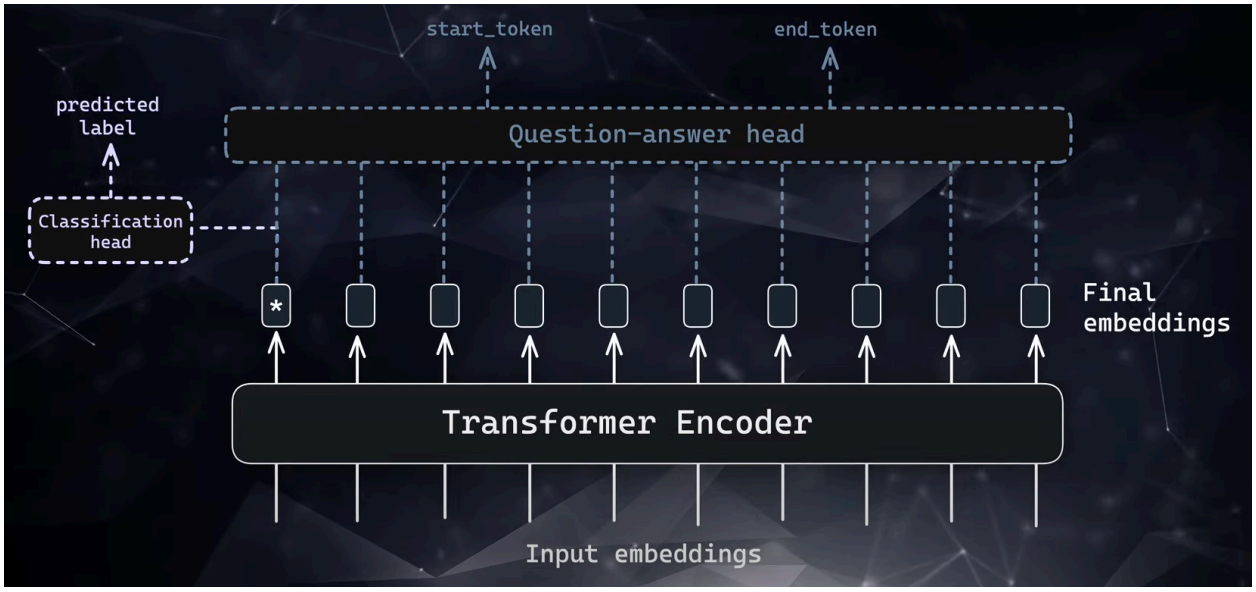
| Consideration | Explanation | Best Suited Tokenizer |
|---|---|---|
| **Language Characteristics** | If the target language has **rich morphology** (complex word forms), subword tokenization can help by breaking words into smaller meaningful units (e.g., morphemes), allowing better handling of various word forms and inflections. | **Subword Tokenization (BPE, WordPiece, SentencePiece)**: Best for morphologically complex languages like Finnish, Turkish, or Arabic, where many variations of words exist. |
| **Model Objectives** | The specific task of the language model (e.g., text generation, translation, summarization) can dictate the **level of granularity** needed in tokenization. For instance, fine-grained tokenization might be better for translation to capture all nuances, while larger tokens might be better for summarization. | **Character-level or Subword Tokenization**: For tasks requiring fine detail (translation), subword tokenization is best; **Word-level Tokenization**: For tasks requiring a broader understanding (summarization). |
| **Computational Efficiency** | For large-scale applications or in **resource-constrained environments** (e.g., mobile devices), computational efficiency is crucial. The choice of tokenizer can impact **model size**, **training speed**, and **inference latency**. Larger token units reduce the sequence length, improving efficiency. | **Word-level or Subword Tokenization**: Reduces sequence length and boosts efficiency in resource-constrained environments. Character-level tokenization may increase sequence length and be less efficient. |
| **Data Availability** | In **data-scarce** scenarios, tokenizers that can **generalize better** from smaller datasets are preferred. Subword tokenization allows breaking rare words into known subword units, improving generalization even when encountering unseen words. | **Subword Tokenization (BPE, WordPiece)**: Ideal for data-scarce scenarios because they generalize well even with small datasets, breaking words into common subunits. |
| **Reversibility** | Some tasks might require easy **reversibility** of the tokenization process (e.g., in text generation tasks). Certain tokenizers, like SentencePiece, which maintain spaces and word boundaries, allow for easy reconstruction of the original text. | **SentencePiece**: Suitable when easy reversibility is necessary, as it retains spaces and word boundaries, making it easier to reconstruct the original text. |
| **Flexibility Across Languages** | For multi-language models or languages with no clear word boundaries (e.g., Chinese, Japanese), a tokenizer must be flexible enough to handle continuous character streams without relying on spaces. | **SentencePiece or Character-level Tokenization**: Best for languages without word boundaries or languages with complex scripts (e.g., Japanese, Chinese), where spaces cannot be relied upon for tokenization. |

- Tokens vs Embeddings
    - Token IDs are a straightforward numerical representation of tokens. It is, in fact, a basic form of vectorization. They do not capture any deeper relationships or patterns between the tokens.
    - Embeddings are advanced vector representations of tokens. They try to capture the most nuance, connections, and semantic meanings between tokens. Each embedding is generally a series of real numbers on a vector space computed by a neural network.
    - In short, text is converted to tokens. Tokens are assigned token IDs. These token IDs can be used to create embeddings for more nuanced numerical representation in complex models.
    - Each token's embedding is a high-dimensional vector. This allows the model to capture a wide range of linguistic features and nuances, like the meaning of a word, its part of speech, and its relationship to other words in the sentence.
    - Contextual Embeddings: Unlike simpler word embeddings (like Word2Vec), BERT's embeddings are contextual. This means the same word can have different embeddings based on its context (its surrounding words). The embeddings need to be rich and complex to capture this contextual nuance.
- Vision transformers
    - Images are broken into smaller patches and flattened into a vector. The spatial arrangement of the vectors are maintained through positional encoders
    - Since attention basically matches everything to everything, we cannot do it for every pixel in an image, hence patches

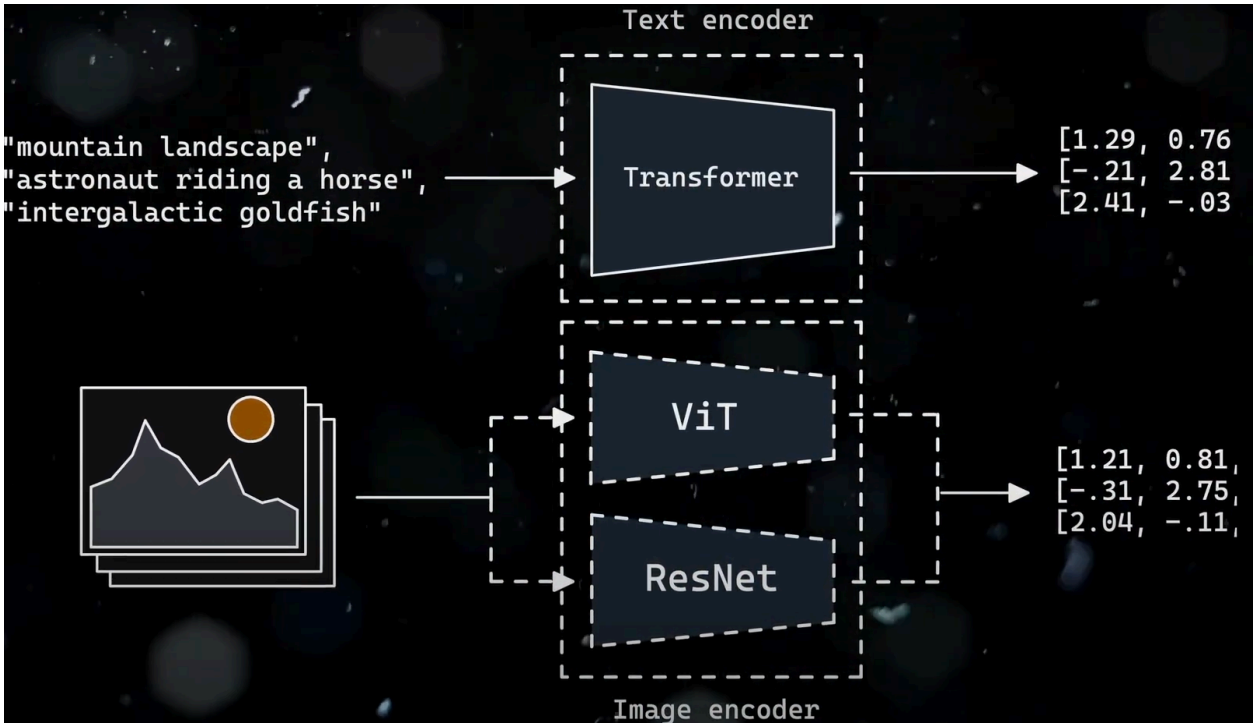


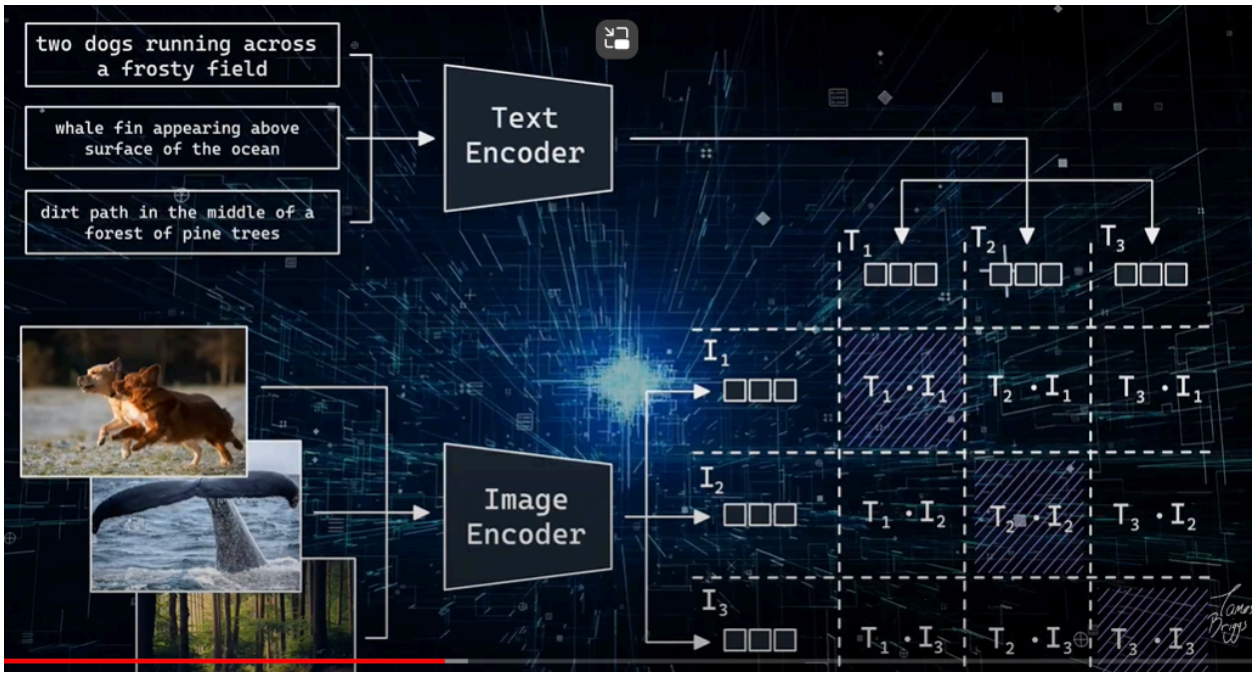

    - We add a QA head, classification head on top of these embeddings to help with the task


Final
embeddings
Transformer Encoder
* Extra learnable
[class] embedding
0 * 1 2 3 4 5 6 7 8 9
Patch + Position
Embedding
Linear Projection of Flattened Patches


start_token
end_token
predicted
label
Question-answer head
Classification
head
*
Final
embeddings
Transformer Encoder
Input embeddings

- Multimodal models
    - CLIP is an example. We train the model to take in an image(two dogs on a frosty field) and text(two dogs on a frosty field) and give vector representations for both to be very close in the vector space.
    - So when we do image to text or text to image searches, we are actually searching in the vector space and not after converting the image to text.




    - Uses contrastive learning to learn. There are positive pairs and negative pairs. Have a loss function that will minimize the difference between the positive pairs

and maximize the difference between the negative pairs. Batch size is an important consideration since if batch size is 2, it will be easy for the model to differentiate which is similar and dissimilar.

- Normal models would have different understanding of the training data but that is where CLIP differs. CLIP training ensures that the image encoder is trained while considering the modality of the text encoder and vice versa. The image and text encoder are kind of sharing an indirect understanding of the others modality.
- CLIP has amazing capability as a model for zero shot tasks. For example for classification tasks, you can convert the target to kind of a sentence and use class embeddings to work in the same way. It can also work well on object detection

## Summary Table: Training Transformers for Text, Audio, Image, and Video

| Aspect | Text | Audio | Image | Video |
|---|---|---|---|---|
| **Preprocessing** | Tokenization into words or subwords | Feature extraction (e.g., spectrograms, raw waveforms) | Split into patches (e.g., 16x16 pixels) | Split into frames (or patches within frames) |
| **Tokenization** | Tokens from words or subwords | Frames or features treated as tokens | Patches treated as tokens | Frames or patches within frames treated as tokens |
| **Positional Encoding** | Temporal (sequence order of tokens) | Temporal (sequence order of audio frames) | Spatial (position of patches in the image) | Spatio-temporal (position within frames and over time) |
| **Transformer Architecture** | Self-attention on tokens | Self-attention on audio frames | Self-attention on image patches | Self-attention on frames or patches over time |
| **Training Objectives** | MLM, CLM, text classification | Self-supervised learning, speech recognition | Image classification, masked image modeling | Action recognition, frame prediction |
| **Challenges** | Relatively short sequences, efficient training | Long sequences, high memory usage | High dimensionality, capturing local and | |

- Tfidf vs tokenizer vs embeddings

| Aspect | TF-IDF | Tokenization | Embeddings (Modern) |
|---|---|---|---|
| **Function** | Statistical measure of term importance in a document | Splits text into words, subwords, or tokens | Dense vector representations that capture semantic relationships |
| **Data Representation** | Sparse vectors with weights for each token | Produces tokens, which can be used by TF-IDF or embeddings | Dense vectors with fewer dimensions compared to TF-IDF |
| **Captures Context/Meaning** | No, focuses on term frequency and document rarity | No, it just splits text | Yes, captures context and meaning through learned vectors |
| **Application** | Information retrieval, document ranking | Preprocessing step in text analysis | NLP tasks like classification, translation, summarization |

- Word2Vec
    - Generating dense word embeddings by training on large corpora of text. It uses a neural network to learn word associations based on their contexts in sentences.
    - Word2Vec generates static embeddings, meaning that each word has a single, fixed vector representation that doesn't change across contexts.
    - Word-2-Vec is based on presence of neighboring words in a moving window fashion. In case of Skipgram, we observe what words appear in a neighborhood of any given corpus word.
    - Word2Vec can be trained using two main methods
        - CBOW (Continuous Bag of Words): Predicts a word given the surrounding context (neighboring words).
        - Skip-Gram: Predicts the surrounding words given a central word.
    - Context-insensitive: Word2Vec cannot handle polysemy (multiple meanings of the same word).
- GloVe (Global Vectors for Word Representation)
    - Method for generating static word embeddings. It combines the strengths of word

co-occurrence statistics and dimensionality reduction.
    - GloVe creates a global co-occurrence matrix, which captures how often words appear together across the entire corpus, and then factorizes this matrix to produce word embeddings.
    - The goal of GloVe is to factorize this matrix to produce word embeddings that capture both local (context-specific) and global (corpus-wide) word relationships.
    - Context-insensitive: Like Word2Vec, GloVe cannot handle polysemy or context-based meanings of words.
    - w3 is relevant to both w1 and w2: $$\frac{P(w_3|w_1)}{P(w_3|w_2)} \approx 1 \qquad (2)$$
    - w3 is only relevant to one of w1 or w2, not both

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \gg 1 \qquad (3)$$

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \ll 1 \qquad (4)$$

    - w3 is irrelevant to both w1 and w2: $$\frac{P(w_3|w_1)}{P(w_3|w_2)} \approx 1 \qquad (5)$$
- BERT Embeddings
    - BERT is a contextual embedding model based on the Transformer architecture. Unlike Word2Vec and GloVe, which produce static embeddings, BERT generates dynamic embeddings that change based on the word's context.
    - Training method:
        - Masked Language Modeling (MLM): During training, some words in a sentence are masked, and BERT learns to predict the masked word based on its context (both left and right contexts).
        - Next Sentence Prediction (NSP): BERT is also trained to predict whether one sentence follows another, improving its understanding of sentence-level relationships.
    - Computationally expensive and Slower than static embeddings

- Hyperparameters
    - Model Hyperparameters: Control the model's architecture and behavior, such as the base LLM model, sequence length, and prompt loss weight configuration.
        - No of layers, no of attention heads, context window, dropout rate, weight decay(penalize large weights to control overfitting)
    - Training Hyperparameters: Control the training process, such as batch size, epoch configuration, and learning rate configuration.

    - Experiment with LoRA Ranks: The LoRA rank controls the number of trainable parameters and model expressiveness. Lower ranks (8-16) are suitable for fine-tuning on a base model, while higher ranks (32-64) are better for teaching new concepts.
    - Balance LoRA Hyperparameters: Adjust the LoRA rank (r) and alpha (scaling parameter) together to find the optimal combination for your model. Alpha is a scaling parameter that controls the strength of the low-rank approximation. A higher alpha value places more emphasis on the low-rank structure, while a lower value reduces its influence.  This balances the pretrained model's knowledge and the new task-specific adaptation
    - Enabling LoRA on more layers allows the model to learn more nuanced representations of the input data. This can lead to:
        - Better performance on the target task
        - Reduced overfitting
        - More effective fine-tuning
    - Temperature is a parameter ranging from 0 to 1, which defines the randomness of LLM responses. The higher the temperature, the more diverse and creative the output would be
    - Top-K sampling: Limits the sampling pool to the top K most probable tokens, reducing the likelihood of generating less relevant or nonsensical text.

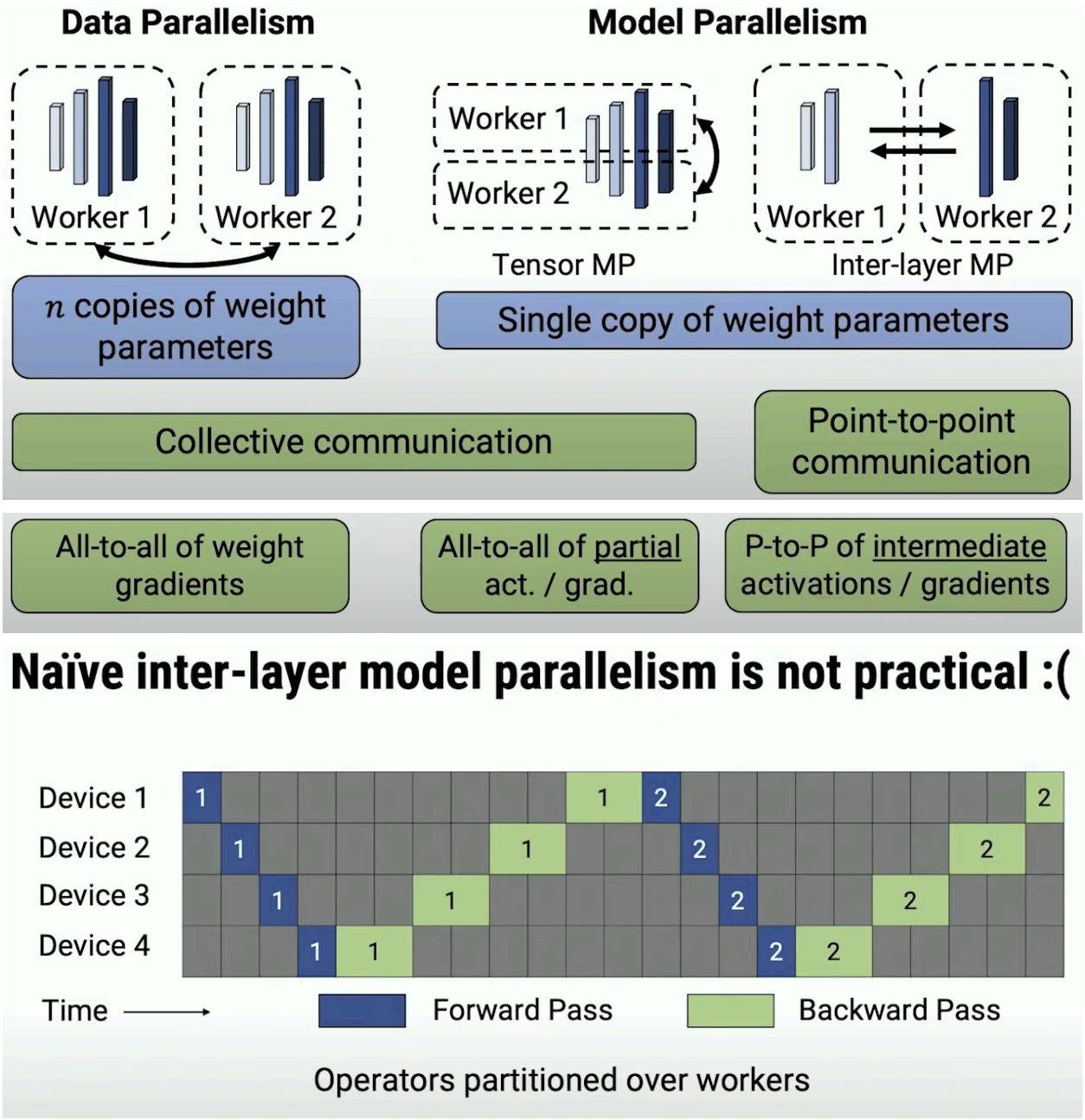
- Metrics
    - Faithfullness: It is useful for measuring if the response was hallucinated and measures if the response from a query engine matches any source nodes.
    - Relevancy:  It is useful for measuring if the query was actually answered by the response and measures if the response + source nodes match the query.
    - Hit Rate: Hit Rate measures the proportion of queries for which the correct chunk/ context appears within the top-k results chunks/ contexts. Put simply, it evaluates how frequently our system correctly identifies the chunk within its top-k chunks.
    - Mean Reciprocal Rank(MRR): MRR assesses a system's accuracy by taking into account the position of the highest-ranking relevant chunk/ context for each query. It calculates the average of the inverse of these positions across all queries. For instance, if the first relevant chunk/ context is at the top of the list, its reciprocal rank is 1. If it's the second item, the reciprocal rank becomes 1/2, and this pattern continues accordingly.
    - Perplexity:  LLM's ability to predict the next word in a sequence. Lower perplexity scores indicate that the model predicts the next word more accurately
    - Accuracy: Accuracy is a widely used metric for classification tasks, representing the proportion of correct predictions made by the model. When generating creative or contextually nuanced text, the "correctness" of the output is not as straightforward.
    - BLEU (Bilingual Evaluation Understudy): Used to evaluate the quality of

generated text by comparing it to reference texts. BLEU is all about precision: if a machine translation uses the exact same words as a human translation, it gets a high BLEU score. For example, if the human reference is "The cat is on the mat," and the machine output is "The cat sits on the mat," the BLEU score would be high because many words overlap.

- ROUGE (Recall-Oriented Understudy for Gisting Evaluation): ROUGE focuses on recall: it checks if the machine-generated text captures all the important ideas from the human reference. Let's say a human-written summary is "The study found that people who exercise regularly tend to have lower blood pressure." If the AI-generated summary is "Exercise linked to lower blood pressure," ROUGE would give it a high score because it captures the main point even though the wording is different.
- Bias and fairness metrics
  - Demographic parity indicates whether a model's performance is consistent across different demographic groups.
  - Equal opportunity focuses on whether the model's errors are evenly distributed across different demographic groups.
  - Counterfactual fairness evaluates whether a model's predictions would change if certain sensitive attributes were different. This involves generating counterfactual examples where the sensitive attribute (e.g., gender or race) is altered while keeping other features constant.
- **Coherence** helps analyze the logical flow and consistency of the generated text. A coherent text maintains a clear structure and logical progression of ideas, making it straightforward for readers to follow.
- **Factuality** evaluates the accuracy of the information provided by the LLM, especially in information-seeking tasks.

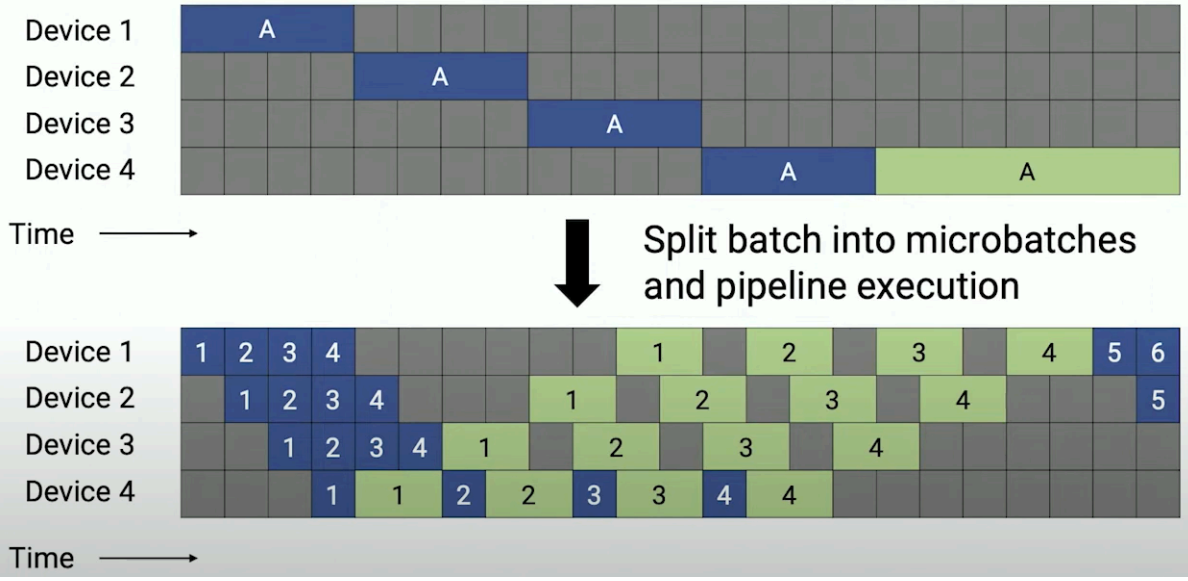
- GPU parallelization
  - Data parallelism: Data Parallelism splits the input data across multiple GPUs. Each GPU has a copy of the full model, and during training, each GPU processes a different portion of the data. After processing, the gradients are averaged across all GPUs, and the model parameters are updated based on the averaged gradients.
    - Inefficient for large models:
  - Model parallelism: Model Parallelism involves splitting the model itself across multiple GPUs. Different parts of the model are placed on different GPUs, and during the forward and backward passes, the intermediate outputs are passed between GPUs.
    - Inter-GPU communication
  - Tensor parallelism: In Tensor Parallelism, the individual tensors (matrices or vectors) of the model are split across GPUs, so that each GPU only holds a portion of the tensor. This approach is commonly used for large matrix multiplications in transformers, such as in the attention layers.
    - Each GPU handles part of the matrix computation, and the results are combined afterward.

        - Tensor parallelism is highly efficient but requires careful tuning to balance load across GPUs and incurs communication overhead
    - Pipeline parallelism: Pipeline Parallelism divides the model into stages, and each stage is assigned to a different GPU. These stages process data in a sequential pipeline, allowing for better memory management.
        - The model is split into different parts (e.g., layers or groups of layers), and each part runs on a different GPU.
        - Idle GPUs and complex scheduling
    - Inter-layer model parallelism
- Useful libraries
    - DeepSpeed: A popular library that supports model, data, tensor, and pipeline parallelism. It offers memory optimization techniques, mixed precision training, and handles large-scale LLM training with ease. DeepSpeed integrates well with PyTorch and Hugging Face Transformers.
    - Megatron-LM: A framework developed by NVIDIA specifically for large-scale transformer models, focusing on tensor parallelism and model parallelism. It's highly optimized for training models with hundreds of billions of parameters.
- Tips and best practices
    - Use mixed precision (FP16) to reduce memory usage and speed up training. Libraries like NVIDIA's Apex and PyTorch AMP (Automatic Mixed Precision) can help with this.
    - DeepSpeed and Megatron-LM also support mixed precision out of the box.
    - During the training of large AI models, activations, or the outputs of different layers in the model, eat up a significant amount of memory.
    - To cut down this large memory requirement, there's a trick called activation checkpointing. This technique reduces the memory used by activations to about a square root of the total activations, but at the cost of having to do about 33% of the calculations again.
    - ZeRO(Zero Redundancy Optimizer): ZeRO achieves significant memory savings by partitioning model states (optimizer states, gradients, and parameters) across multiple GPUs during data parallelism (DP), allowing larger models to be trained efficiently on available hardware. Has 3 stages optimizer, gradient, parameter. Can also offload these to disk to free up GPU memory.

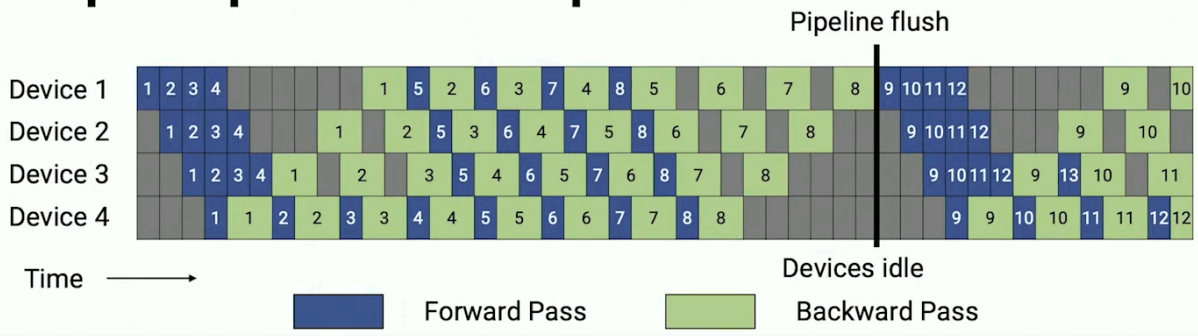



# Naïve inter-layer model parallelism is not practical :(

# Idea: Can we use pipelining to increase throughput?




# Pipeline parallelism: deep dive




Forward time per microbatch $= t_f$

Backward time per microbatch $= t_b$

Size of bubble $= (p-1) \cdot (t_f + t_b)$

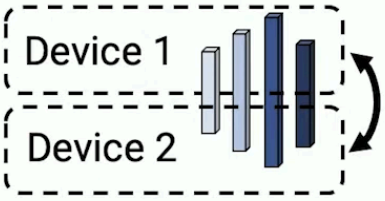
Ideal total time $= m \cdot (t_f + t_b)$

Fraction of time spent in bubble $= \frac{p-1}{m}$

**At large batch sizes, pipeline bubble less expensive**

# Different schemes have different tradeoffs

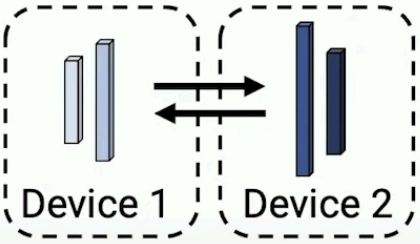
**Tensor Model Parallelism**




**Communication expensive**

**Devices never idle**

**Pipeline Model Parallelism**




**Communication cheap**

**Devices often idle**

# How do we navigate this configuration space?

- Degree of pipeline, tensor, and data parallelism
- Pipelining schedule
- Global batch size
- Microbatch size

} Each of these choices influence amount of communication, size of pipeline bubble, memory footprint