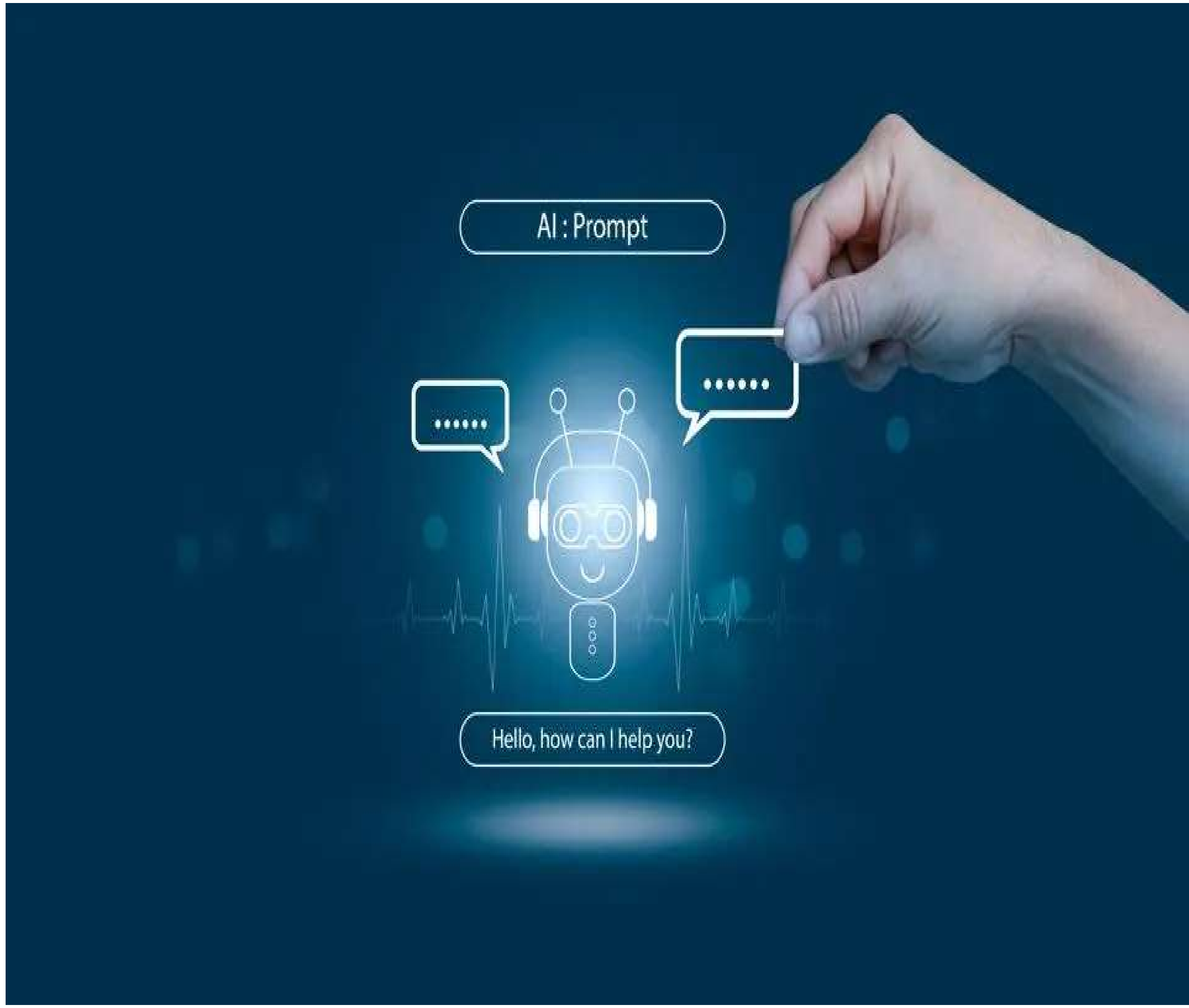
Transformers
LLMs
RAG



Arun sharma K

15/09/2024

Elements of Transformers

Transformers

Encoder

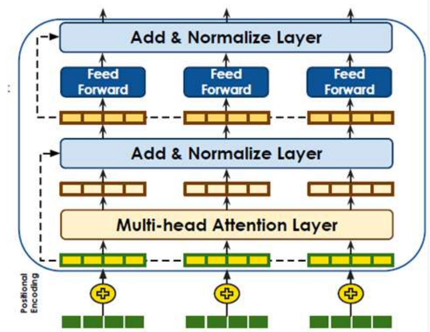
- Word embedding
- Position encoding
- Multi-headed self-Attention
- Skip Connections and Layer Normalization
- Feed forward

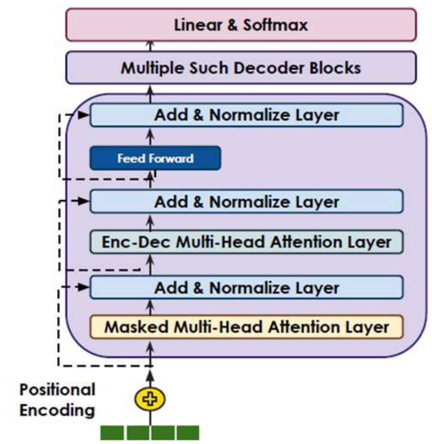
Processes the input sequence and encodes its representation

Decoder

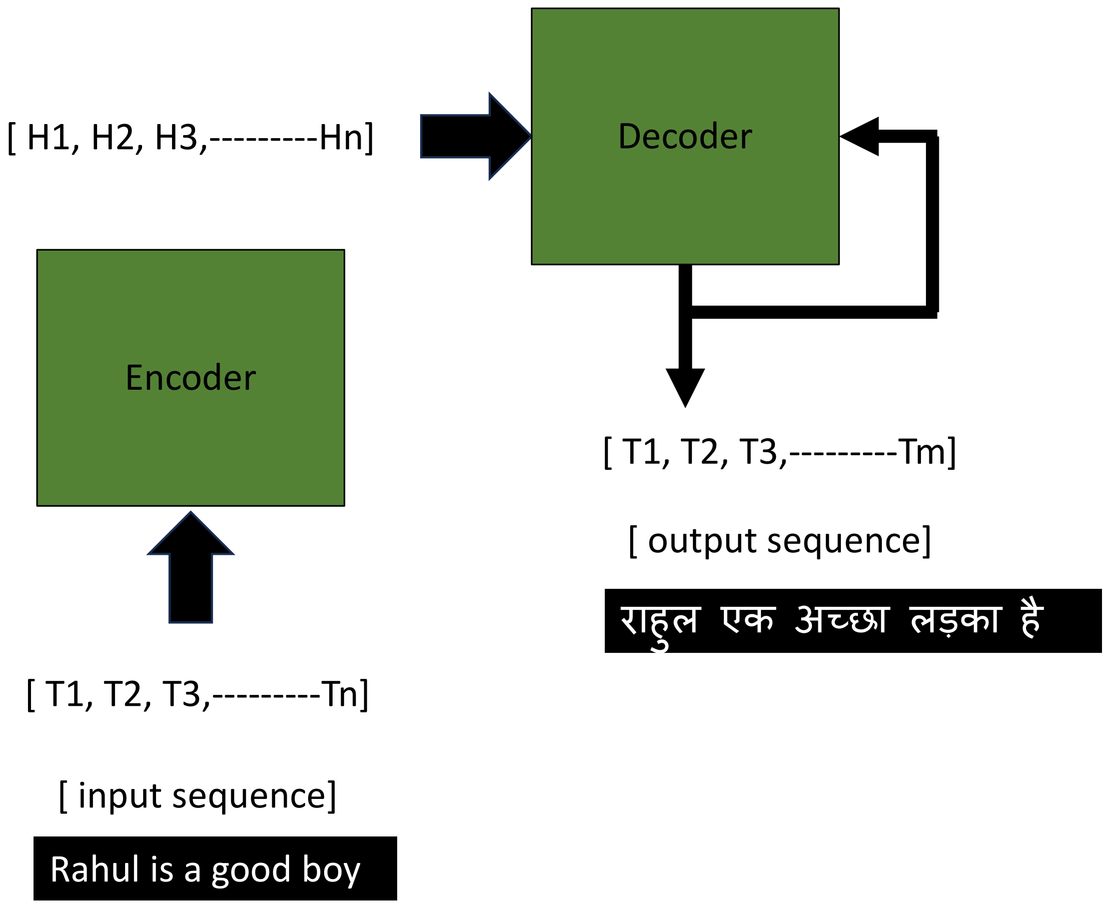
- Position encoding
- Masked Multi-headed attention
- Encoder-Decoder attention
- Skip Connections and Layer Normalization
- Linear and SoftMax

Generates the output sequence (such as the translated sentence) by attending to both the encoded input and the previously generated output





Encode-Decoder



Transformer types

Transformers

Encoder-Decoder

- BART, M2M, mBART

Bidirectional and Auto-Regressive Transformers

machine translation, summarization, and text generation

Encoder only

- BERT

Bidirectional Encoder Representations from Transformers

Semantic analysis, NER, SA, Semantic Search, classification

Decoder only

- GPT*

Text generation, Chatbot, Code generation

Multi-headed self-attention

The meaning of the word depends on words that may or may not be in the immediate neighbourhood

Self-Attention is a way to determine this dependency

One of the core elements of Transformers is Multi-headed Self-attention

Amit love supercars. He is driving a Bugatti

Each word will look at what word is relevant to it

As part of this exercise, multiple relationships might stem out

Each Self-attention captures one of such relations

Multi-headed attention captures multiple relations

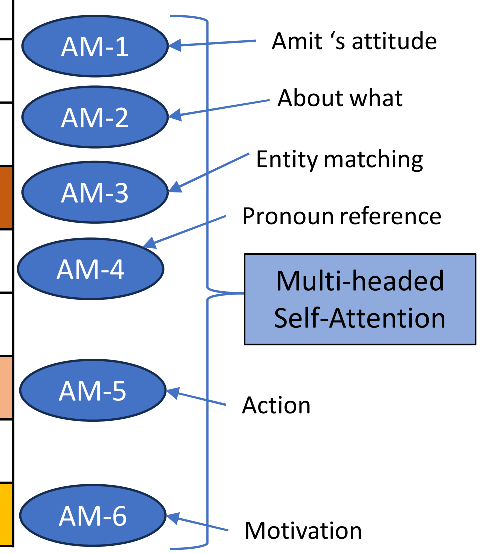
Multi headed self-attention

The meaning of the word depends on words that may or may not be in the immediate neighbourhood

Self-Attention is a way to determine this dependency

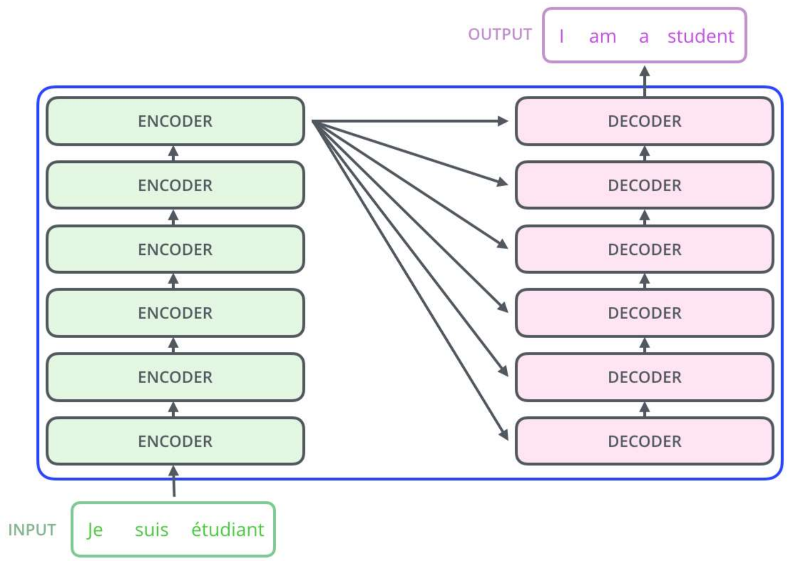
One of the core elements of Transformers is Multi-headed Self-attention

	amit	loves	supercars	he	is	driving	a	bugatti
amit								
loves								
supercars								
he								
is								
driving								
a								
bugatti								



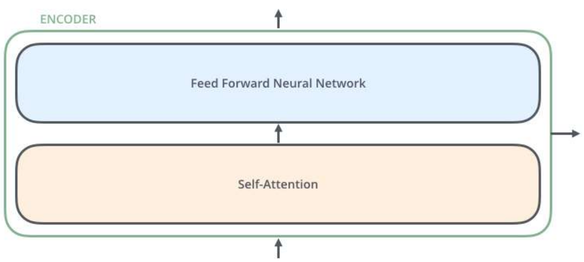
Amit is driving a Bugatti since he loves supercars

Transformers

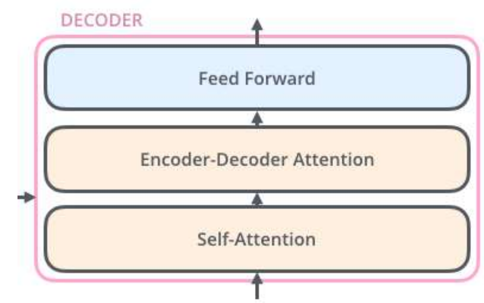


All encoders have the same architecture.

All decoders have the same architecture.

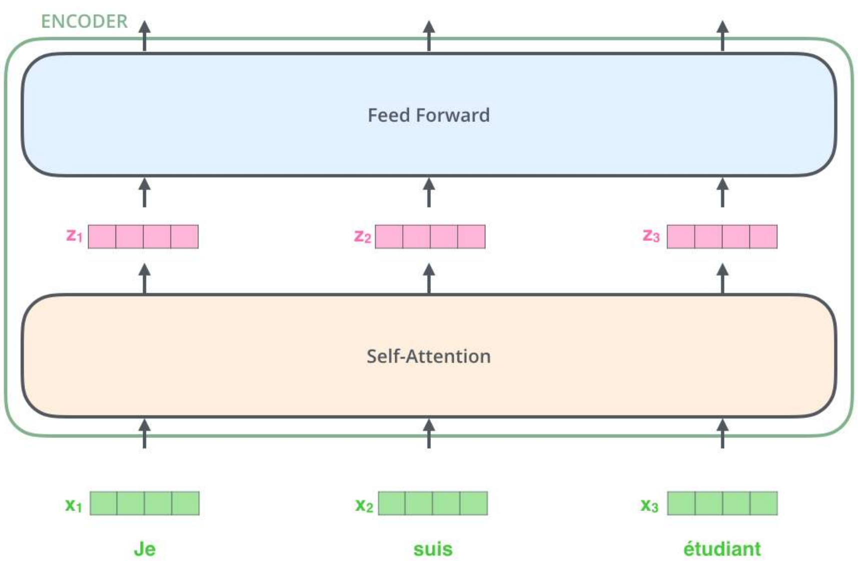


The encoder’s inputs first flow through a **self-attention** layer. It helps the encoder look at other words in the input sentence as it encodes a specific word



The decoder has SA and FF and also , a layer between them: an attention layer that helps the decoder focus on relevant parts of the input sentence

Self Attention



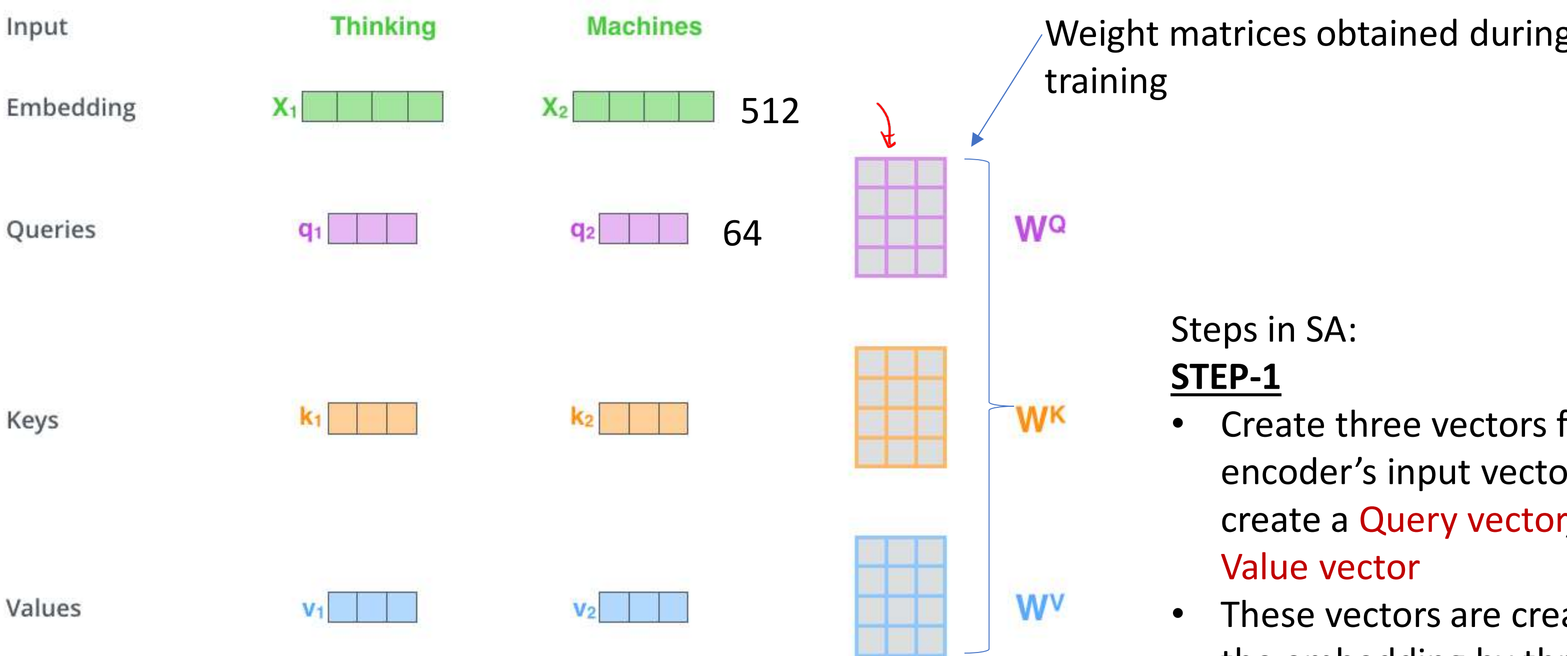
Each word is embedded into a vector of size 512

One key property of the Transformer, which is that the word in each position flows through its own path in the encoder {positional encoding}, dependencies captured only in SA.

Steps in SA:

- Create three vectors from each of the encoder's input vectors. For each word, we create a Query vector, a Key vector, and a Value vector
- These vectors are created by multiplying the embedding by three matrices that we trained during the training process

Self Attention

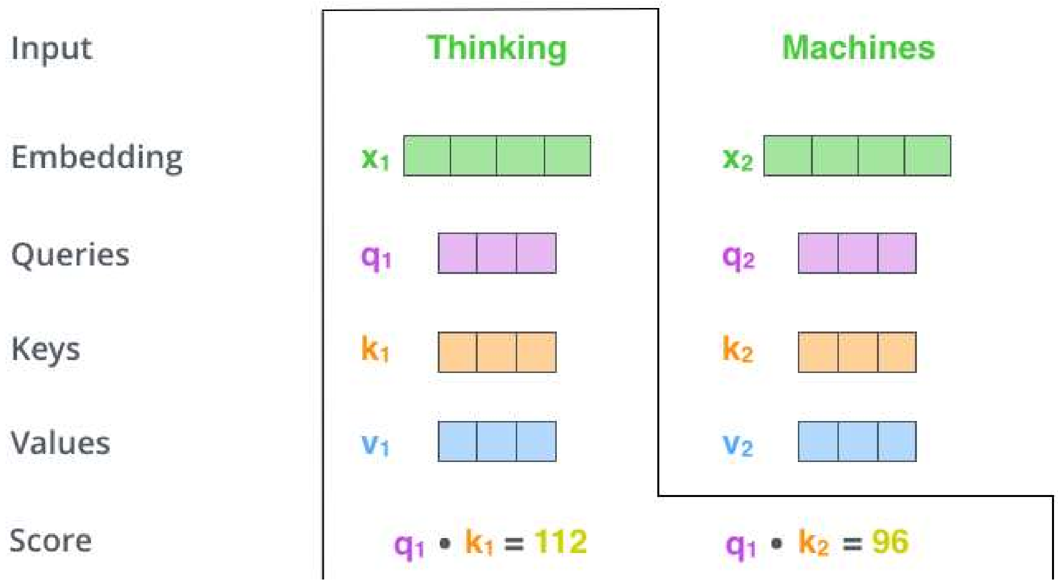


Steps in SA:

STEP-1

- Create three vectors from each of the encoder's input vectors. For each word, we create a Query vector, a Key vector, and a Value vector
- These vectors are created by multiplying the embedding by three matrices that we trained during the training process

Self Attention

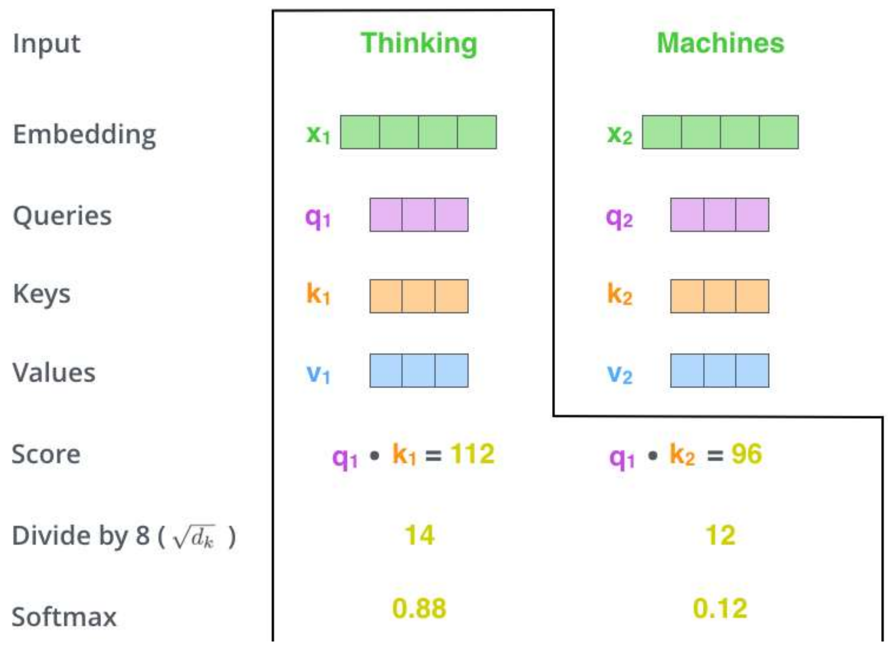


Steps in SA:

STEP-2

- Calculate scores for each word input against the other words. The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position
- The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring

Self Attention



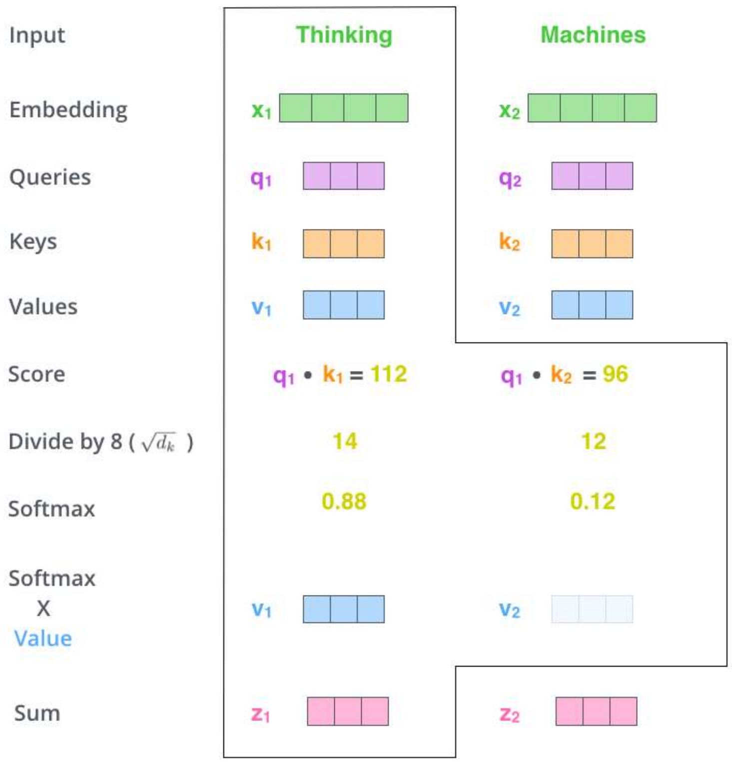
Steps in SA:

<u>**STEP-3 and 4**</u>

- Divide by 8
- Pass the result through a SoftMax operation. SoftMax normalizes the scores so they're all positive and add up to 1

This SoftMax score determines how much each word will be expressed at this position. Clearly the word at its position will have the highest SoftMax score, but it's useful to attend to another word that is relevant to the current word

Self Attention



Steps in SA:

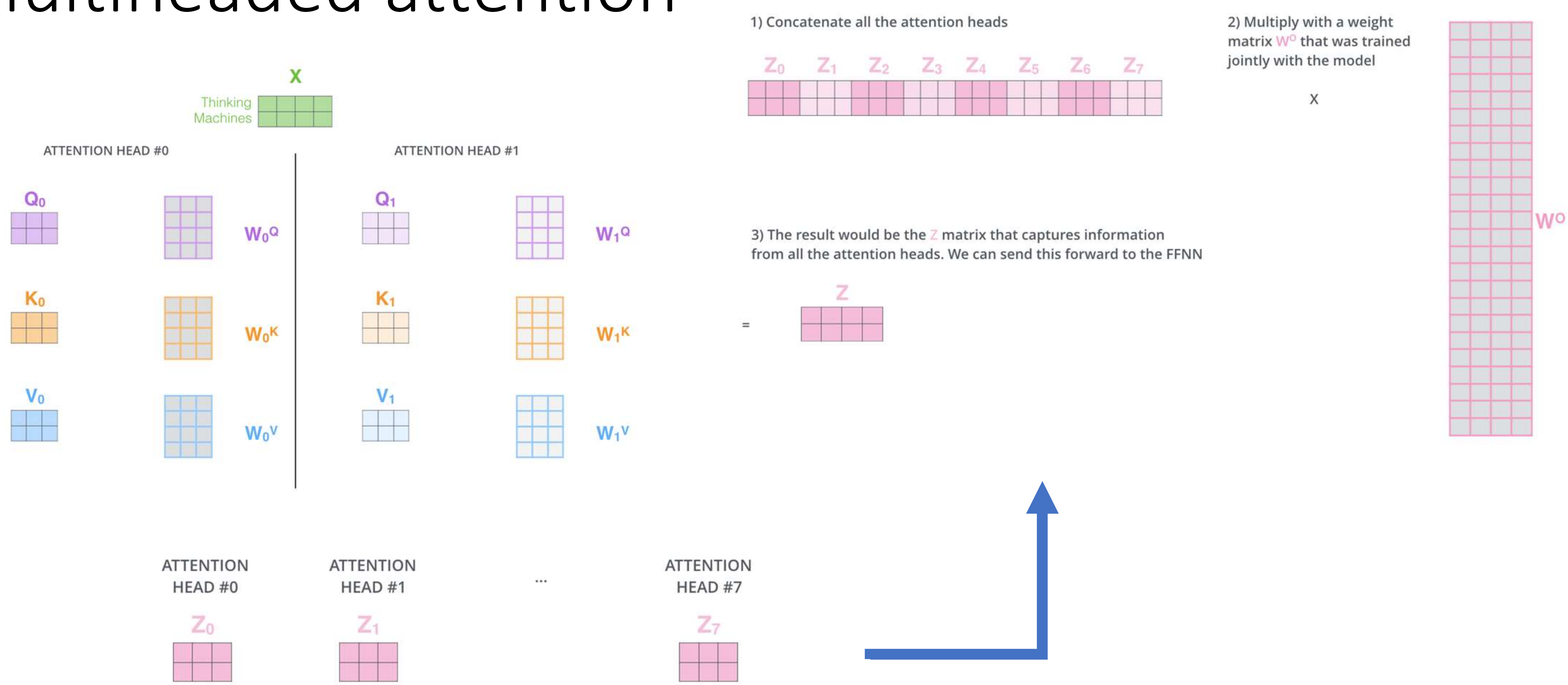
STEP-5 and 6

- multiply each value vector by the SoftMax score. The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words

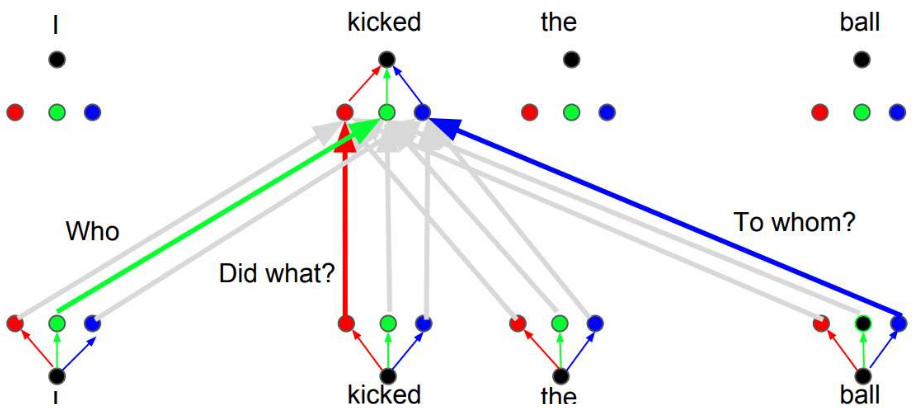
- Sum up the weighted value vectors. This produces the output of the self-attention layer at this position

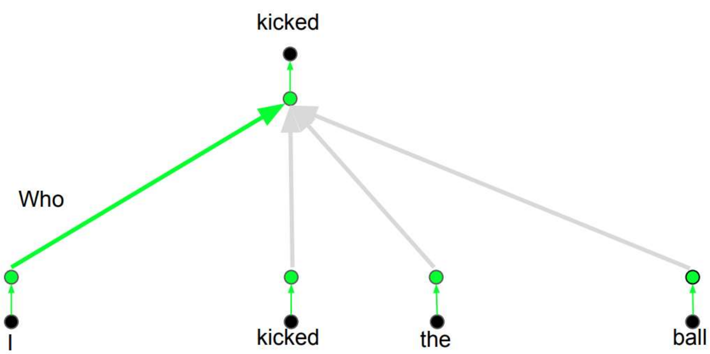
This SoftMax score determines how much each word will be expressed at a position. Clearly the word at its own position will have the highest SoftMax score, but it's useful to attend to another word that is relevant to the current word

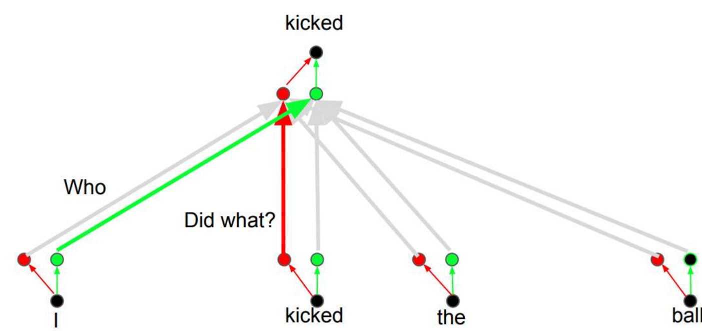
Multiheaded attention



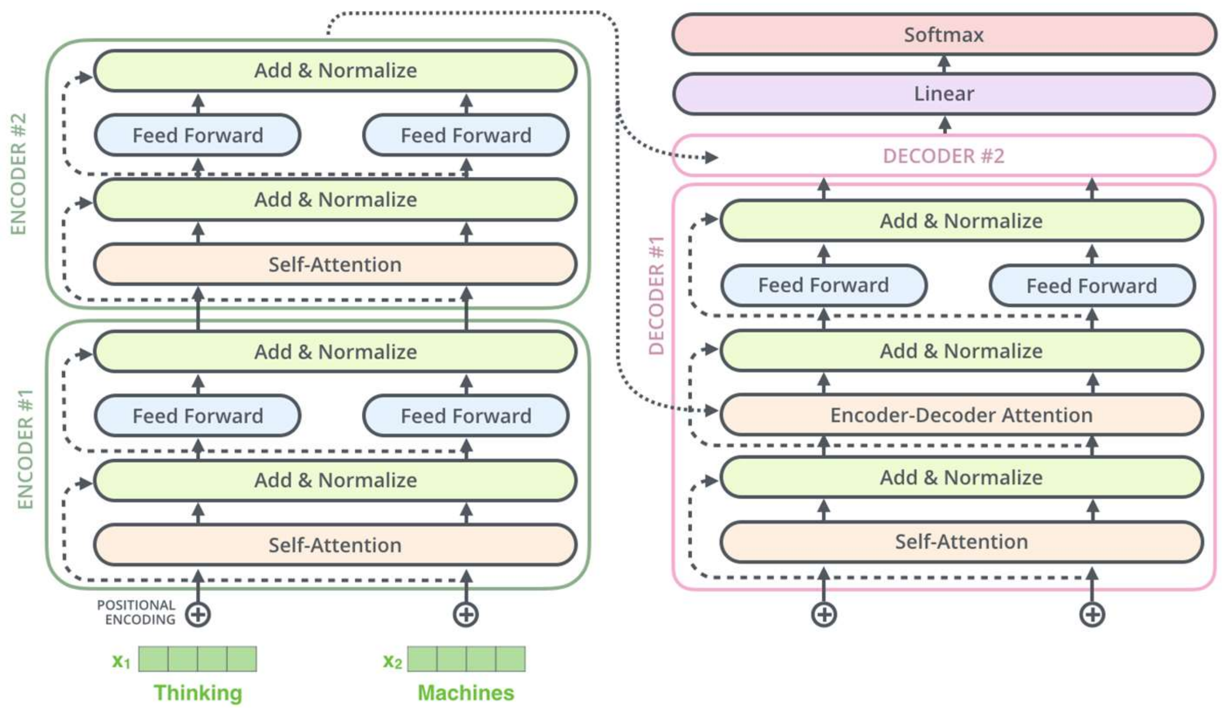
Multi-head attention



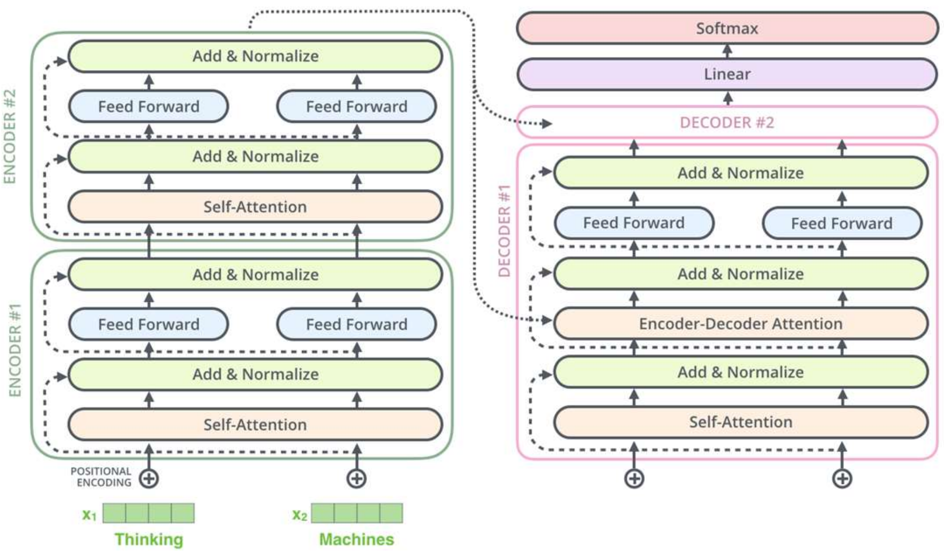




Transformer: more details



Transformer: positional encoding



- https://d2l.ai/chapter_attention-mechanisms/self-attention-and-positional-encoding.html#subsec-positional-encoding

10.6.3. Positional Encoding

Unlike RNNs that recurrently process tokens of a sequence one by one, self-attention ditches sequential operations in favor of parallel computation. To use the sequence order information, we can inject absolute or relative positional information by adding *positional encoding* to the input representations. Positional encodings can be either learned or fixed. In the following, we describe a fixed positional encoding based on sine and cosine functions [Vaswani et al., 2017].

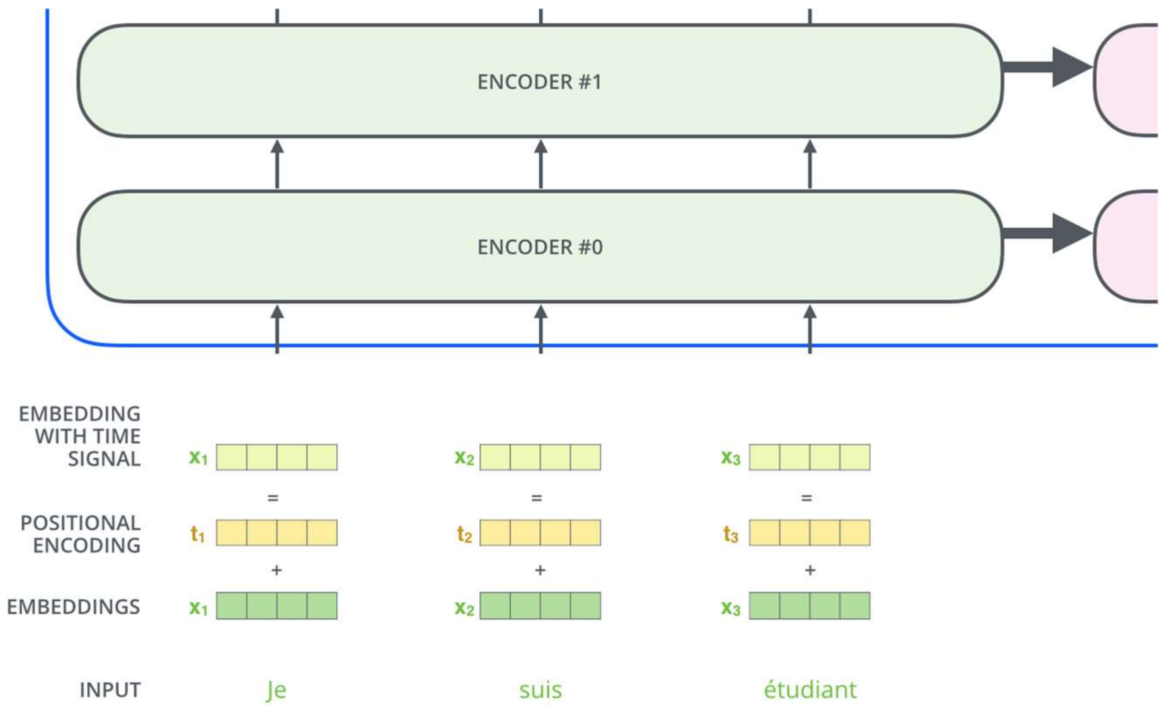
Suppose that the input representation $\mathbf{X} \in \mathbb{R}^{n \times d}$ contains the d-dimensional embeddings for n tokens of a sequence. The positional encoding outputs $\mathbf{X} + \mathbf{P}$ using a positional embedding matrix $\mathbf{P} \in \mathbb{R}^{n \times d}$ of the same shape, whose element on the i^{th} row and the $(2j)^{\mathrm{th}}$ or the $(2j+1)^{\mathrm{th}}$ column is

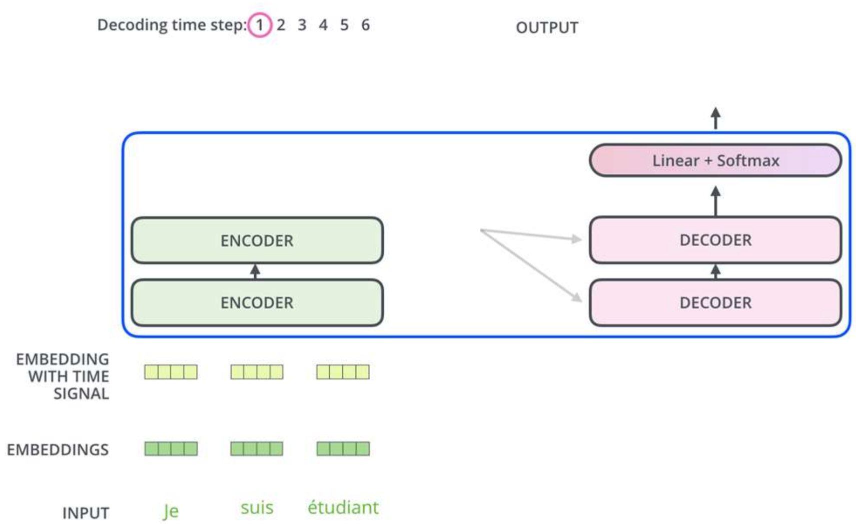
$$\begin{aligned} p_{i,2j} &= \sin\left(\frac{i}{10000^{2j/d}}\right),\\ p_{i,2j+1} &= \cos\left(\frac{i}{10000^{2j/d}}\right). \end{aligned} \tag{10.6.2}$$

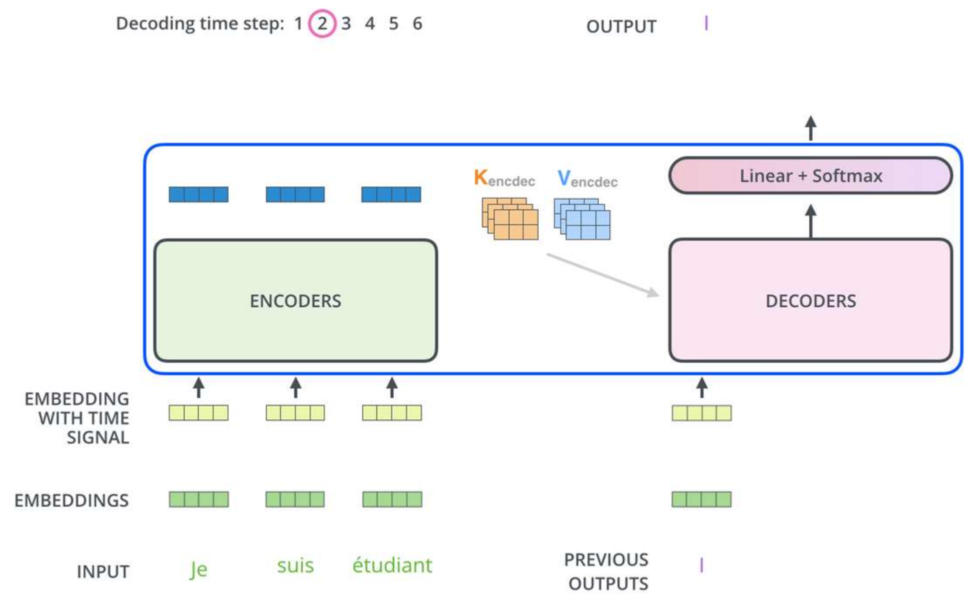
At first glance, this trigonometric-function design looks weird. Before explanations of this design, let us first implement it in the following `PositionalEncoding` class.

Besides capturing absolute positional information, the above positional encoding also allows a model to easily learn to attend by relative positions

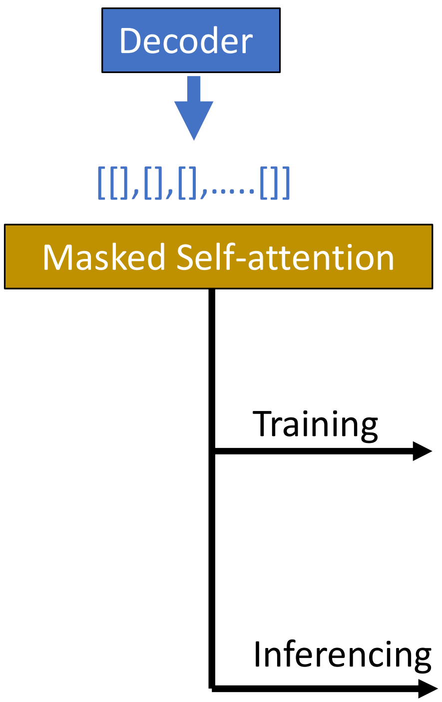
Transformer: positional encoding




Decoding time step: 1 2 3 4 5 6
OUTPUT
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Kencdec
Vencdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Masked Multi headed self-attention



Causal mask (autoregressive mask)

- **Use:** Autoregressive models (Sequential prediction)→ One token at a time.
- **Need**: Capture rich contextual information by mapping dependencies between past and currently predicted tokens
- **Work**: Similar to Self-attention with masking of tokens

- The causal mask is applied to ensure that each token can only "see" the previous tokens since the entire sequence is available during training, This keeps the model from cheating

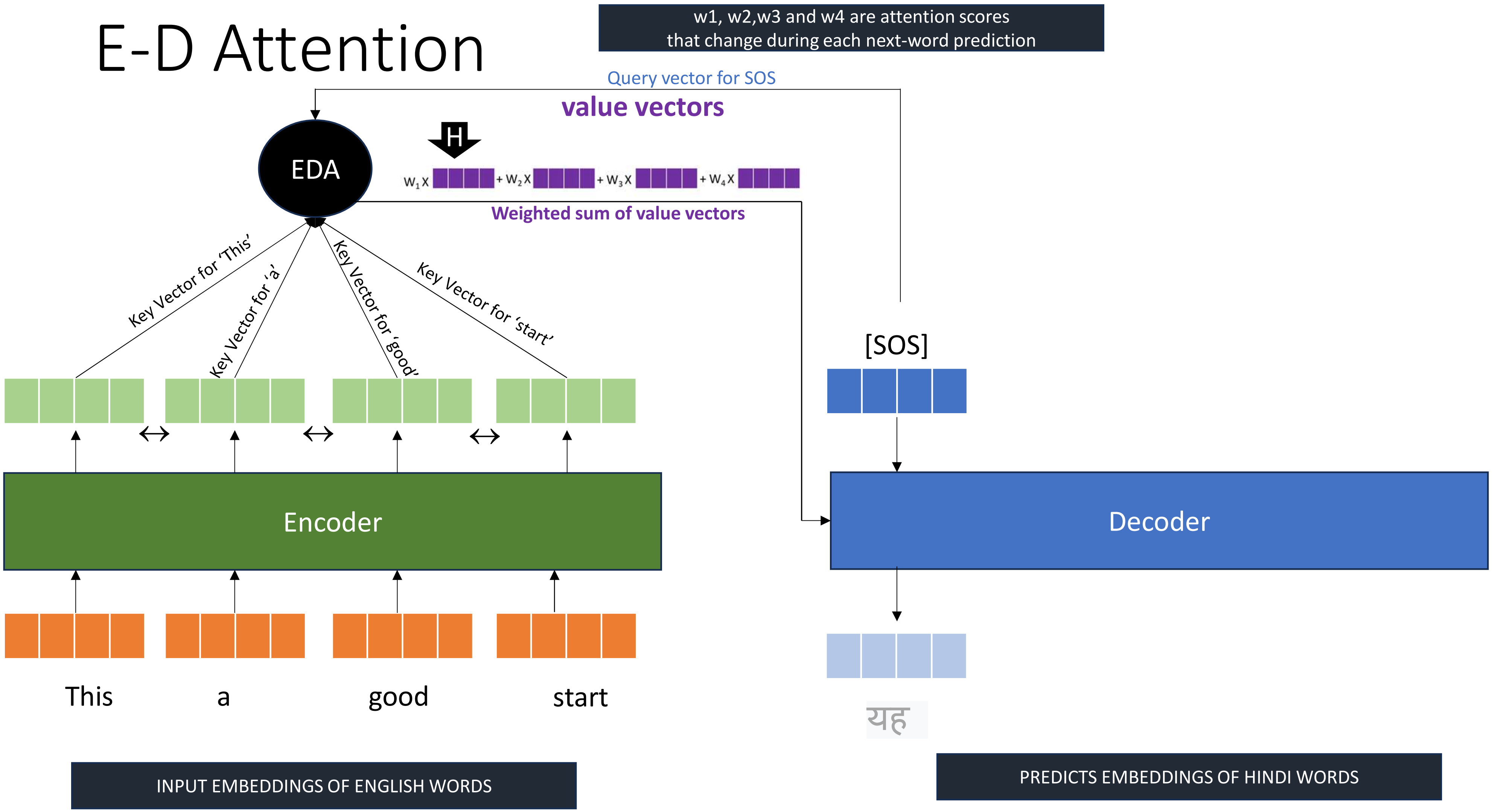
- Masking is turned off in the last layer. The other layers still see masking to maintain consistency between training and inferencing

Masked Multi headed self-attention

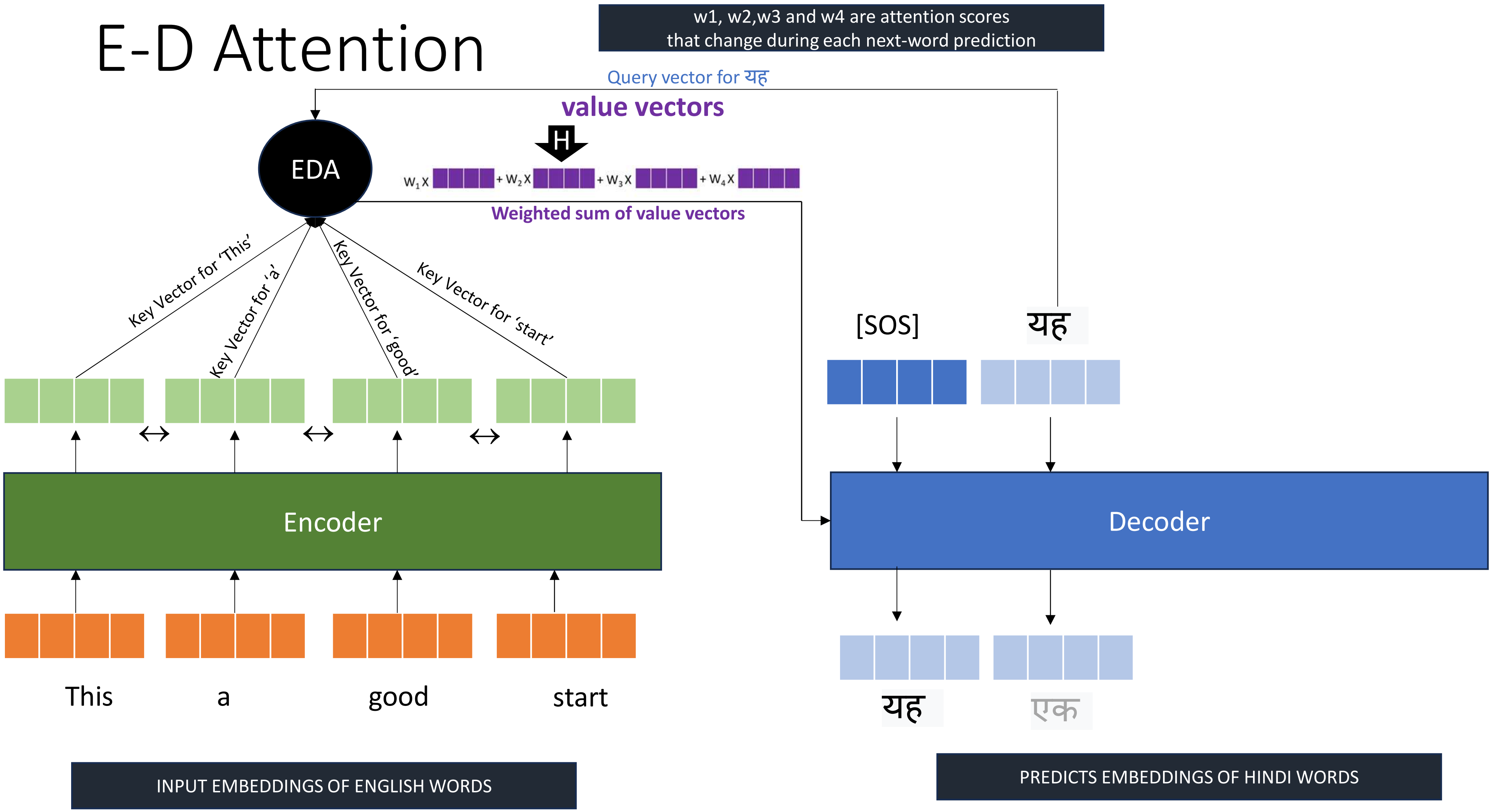
Example of masking:

"It is going to rain heavily today"	Token to be trained as a response
"It is"	Currently predicted tokens
[x.xx, y.yy, -10^8, -10^8, -10^8, -10^8, -10^8]	Masked matrix for predicting the next word ("going")→ Q.K output
[0.65, 0.35, 0, 0, 0, 0, 0]	Attention scores

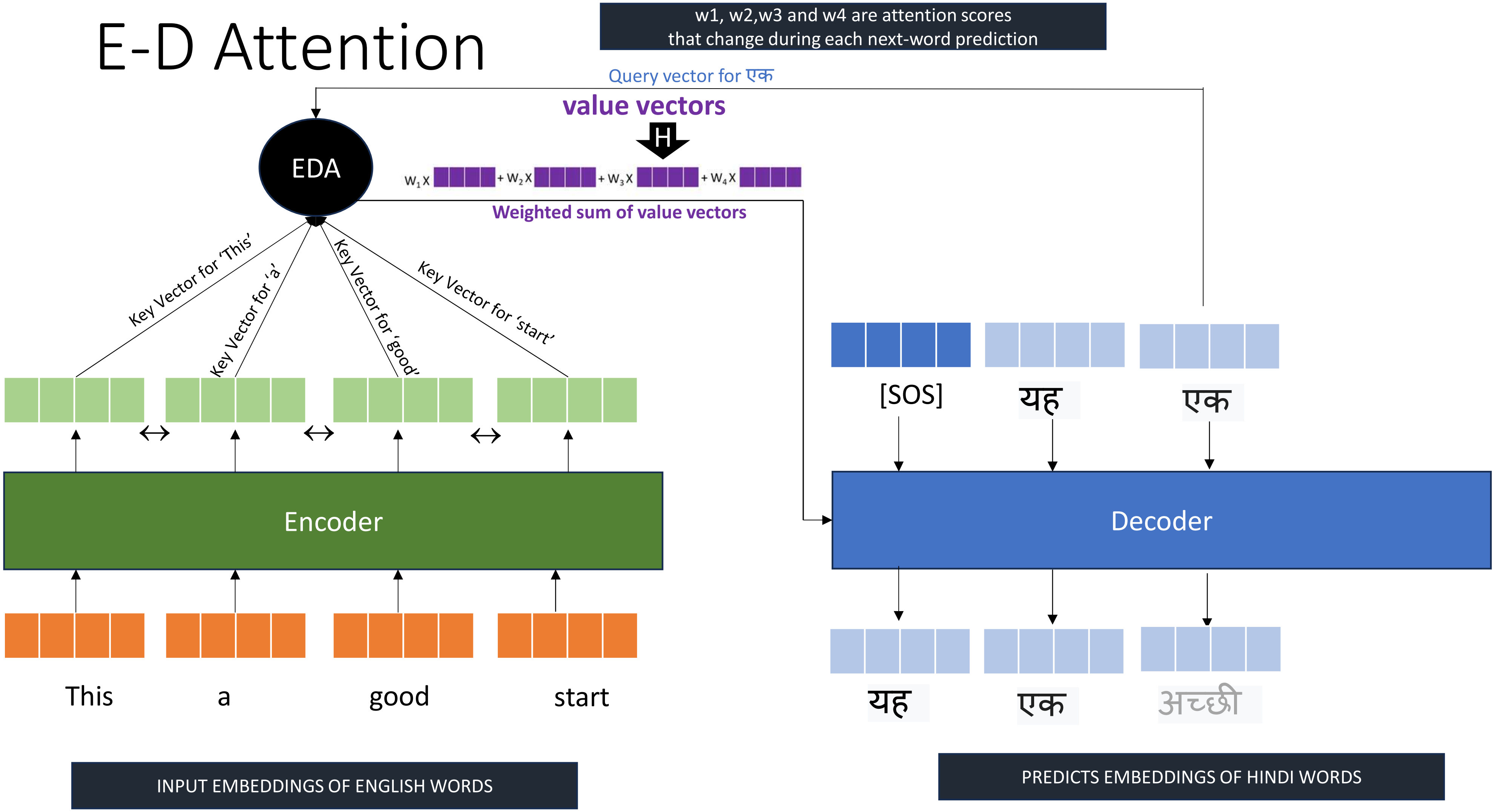
E-D Attention



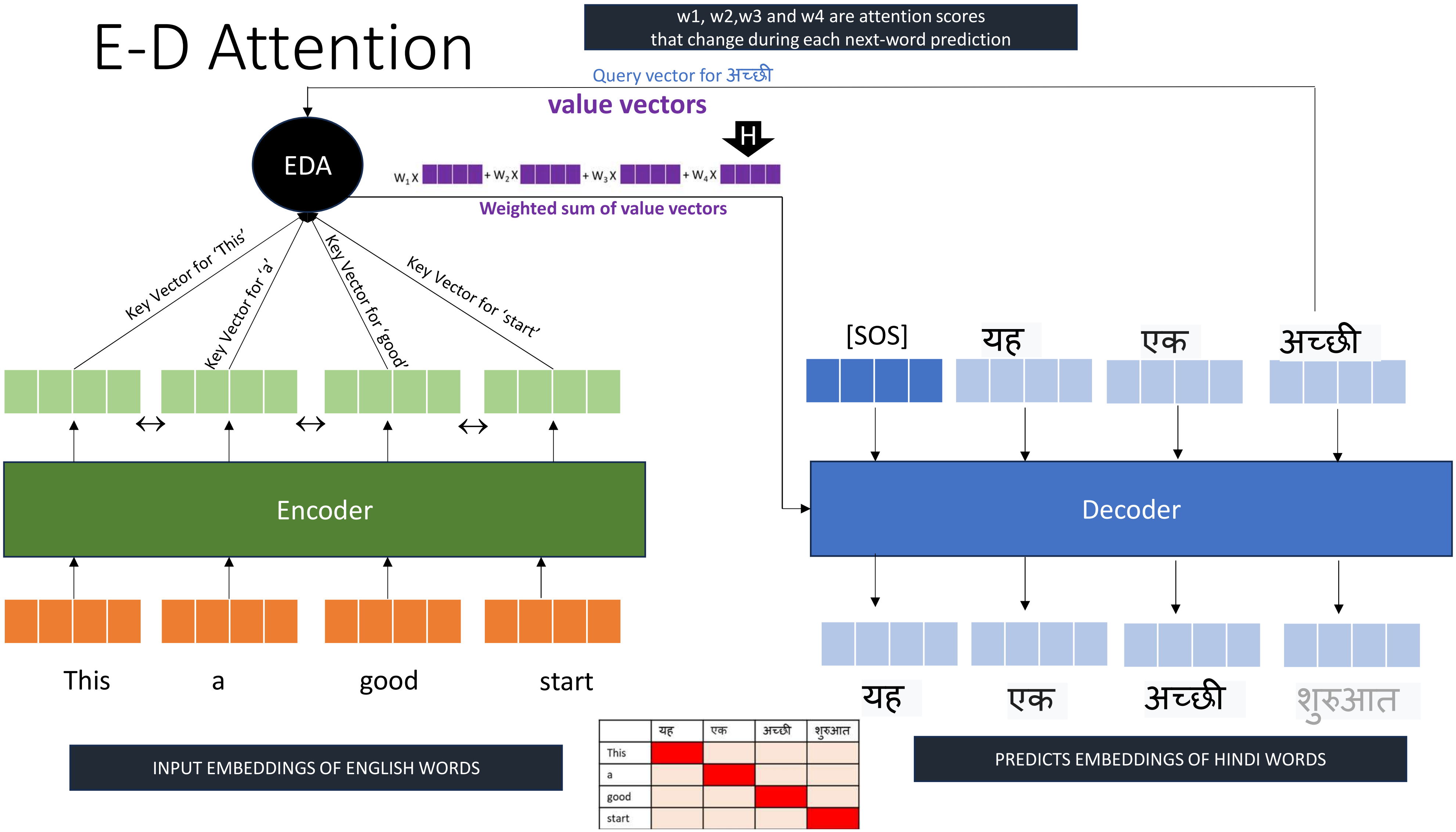
E-D Attention



E-D Attention



E-D Attention



Large Language Models

Large Language Models

Capable of understanding and generating human language

- Pre-trained on a large corpus of text data
- Learn the complex patterns and rules of human language

task automation, customer service chatbots, subject research, document summarization and content generation (text, images, audio, video, code)

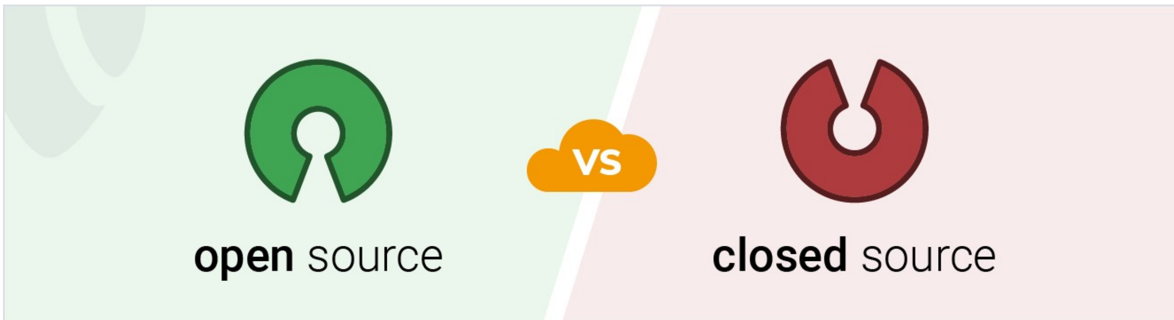
TYPE

Open-source proprietary

pricing, parameter size, context window, customization options, deployability

Web Interface, API Interface, Third-Party Platforms

Popular LLMs



1. GPT-NeoX
2. BLOOM
3. LLaMA 2
4. BERT
5. XGen-7B
6. Falcon-180B
7. Vicuna-33B
8. Dolly 2.0
9. CodeGen
10. latypus 2

Training Large Language Models

The model learns general language patterns, grammar, facts, reasoning abilities, and world knowledge

Base LLM

Large data collection → Pre-train model → Supervised fine-tuning → RLHF → Fine-tuned LLM

books, websites, scientific papers, Wikipedia, news articles, emails, internal documents

Task-specific high-quality human-labeled data

(optional)

Response ranking based on human annotation

self-supervised learning

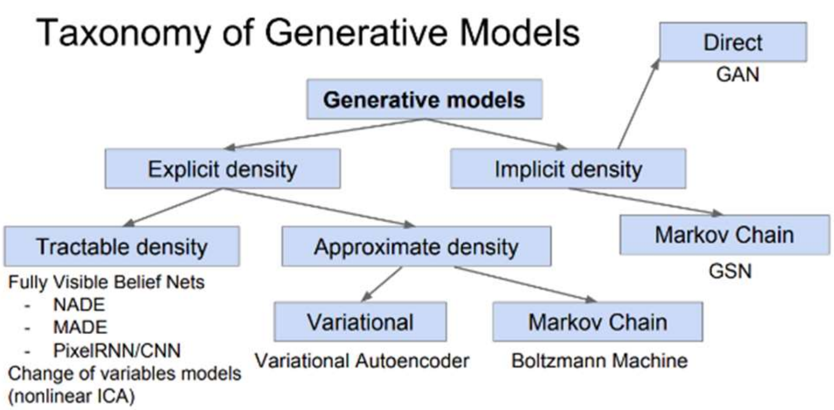
Training objective

Autoregressive Modeling (Models like GPT):
Train the model to predict the next token given the previous tokens. (causal masking)

Masked Language Modeling (BERT-like Models):
Randomly mask some tokens in the input and train the model to predict the masked tokens. This forces the model to learn the context and relationships between words.

Generative Vs Discriminative

	Generative Models	Discriminative Models
Focus	Joint probability distribution P(x, y)	Conditional probability P(y|x)
Goal	Model the underlying data distribution	Learn a decision boundary
Learning Process	Maximize likelihood	Minimize classification error
Applications	Data generation, anomaly detection	Classification, regression



Generative models model $P(x, y)$, which can be factored as:

$$P(x, y) = P(x)P(y|x)$$

or

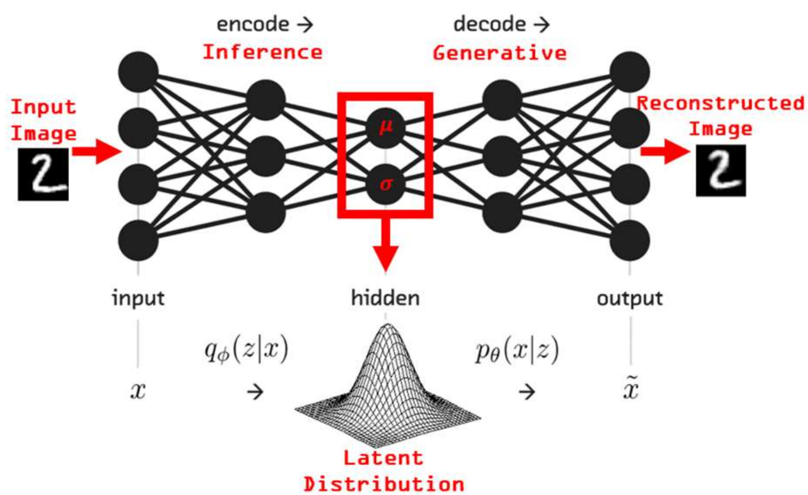
$$P(x, y) = P(y)P(x|y)$$

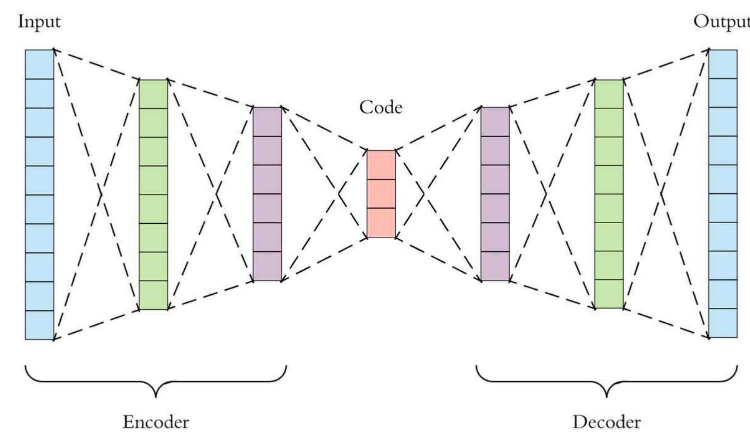
$P(x)$: Prior probability of data.

$P(y|x)$: Conditional probability of labels given data.

$P(x|y)$: Likelihood of data given labels.

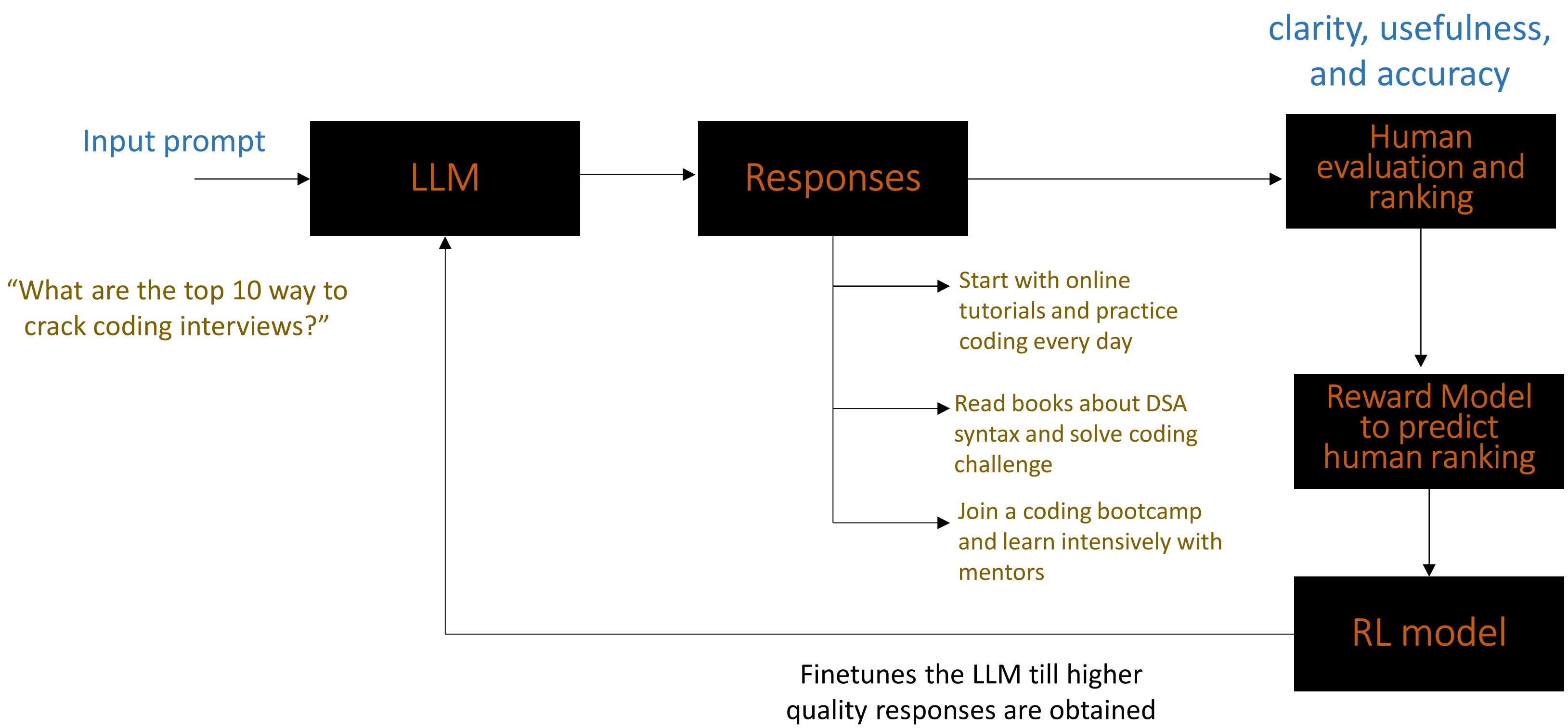
Generative models





Training Large Language Models

Response ranking based on human annotation



Proximal Policy Optimization

Frame works for building Applications

LangChain

LangServe

LangSmith

LangGraph

MetaGPT

MetaGPT (**multi-agent collaboration for complex problem-solving tasks)**. utilizes multiple **intelligent agents**, each with a specialized role, to simulate team-like collaboration, mimicking the way human teams work together.

Frameworks for building LLMs

Feature	LangChain	LangGraph
Workflow Structure	Linear or sequential chains	Graph-based, non-linear workflows
Task Management	Sequential execution of tasks	Flexible execution with nodes and edges
Complexity	Suited for simple and moderately complex tasks	Ideal for complex, interconnected workflows
Dynamic Decisions	Limited support for dynamic branching	Supports conditional and branching logic
Scalability	Moderate scalability	High scalability for multi-agent systems
Primary Use Cases	Single-agent chatbots, RAG systems	Multi-agent collaboration, dynamic pipelines
Multi-Agent Systems	Limited support	Built-in support for multi-agent workflows
Reusability	Limited modularity	High modularity with reusable nodes
Integration	Integrates tools (LLMs, APIs, databases)	Integrates tools with complex dependencies
Key Strengths	Simplifies LLM-driven pipelines	Handles sophisticated, dynamic dependencies
Execution Model	Tasks execute sequentially	Tasks execute based on graph dependencies
Learning Curve	Easier to learn for beginners	Requires understanding of graph structures
Flexibility	Good for most LLM-powered applications	Ideal for highly dynamic and non-linear systems

Vector databases- Options

Database/Library	Description	Features	Use Cases
Redis (Vector Module)	In-memory database with vector capabilities	Real-time queries, Redisearch for hybrid queries, widely used	Real-time recommendations, fast hybrid search
Qdrant	AI-oriented vector database	Developer-friendly, filtering, scoring, local and cloud deployments	Multimedia search, similarity matching, recommendations
ElasticSearch (k-NN)	Search engine with vector search plugin	Combines text and vector search, scalable, k-NN plugin for similarity queries	Hybrid search, recommendations
FAISS	Similarity search library	High performance, supports CPU/GPU, efficient on large datasets	R&D, prototyping, academic use
Annoy	Lightweight search library	Memory-efficient, designed for static datasets, high speed	Music recommendation, small-scale vector search
Pinecone	Fully managed vector database	Scalable, serverless, high-dimensional vector search, ML workflow integration	NLP, recommendations, image/document search
Weaviate	Open-source vector search engine	Hybrid search, schema-free ingestion, integrates with OpenAI/Cohere, graph relationships	Semantic search, knowledge graphs, AI search
Milvus	Cloud-native, open-source vector database	Distributed deployment, supports multiple distance metrics, integrates with ML frameworks	Image/video search, recommendations, autonomous driving
Chroma	AI-focused vector database	Fast prototyping, LLM integration, high-performance querying	Conversational AI, document embeddings
Vespa	Real-time search engine	Integrates text/structured/vector search, customizable, supports ANN search	E-commerce search, analytics, personalization

Vector databases-Tasks

Task	Description	Examples
Similarity Search	Finding vectors in the database closest to a given query vector based on similarity metrics.	Searching for similar documents, recommending products, or finding visually similar images.
Approximate Nearest Neighbour (ANN)	Efficiently finding nearest neighbors for high-dimensional data, trading off exact accuracy.	Real-time recommendation systems, image deduplication, multimedia retrieval.
Hybrid Search	Combining vector similarity search with traditional queries or metadata filtering.	Searching for documents by filters like publication year, or products within a price range.
Clustering	Grouping vectors into clusters based on similarity to identify patterns or categories.	Segmenting customer profiles, identifying communities, grouping similar documents.
Semantic Search	Enabling searches that rely on the meaning of queries rather than exact matches.	Retrieving documents answering user queries, searching images based on text descriptions.
Recommendation Systems	Providing personalized recommendations by comparing user or item embeddings.	Suggesting movies, books, or music; recommending products based on user history.
Anomaly Detection	Identifying vectors that deviate significantly from the norm in a dataset.	Detecting fraud in transactions, spotting unusual user behavior, identifying outlier images.
Data Exploration and Visualization	Using dimensionality reduction to visually explore high-dimensional data.	Analyzing cluster distributions, visualizing relationships between data points.
Multi-Modal Search	Handling searches across different data types like text, images, and audio.	Searching for images similar to text descriptions, or videos matching audio queries.

Vector databases-Tasks

Task	Description	Examples
Vector Ingestion	Adding vector embeddings and associated metadata to the database.	Uploading image embeddings or document embeddings with metadata.
Updating and Deleting Vectors	Modifying or removing embeddings and their metadata.	Updating embeddings for products or removing outdated data points.
Real-Time Querying	Handling live, low-latency vector searches for instant responses.	Recommending products in real-time or matching live queries with stored embeddings.
Ranking and Scoring	Assigning relevance scores to vectors based on similarity to the query and metadata constraints.	Ranking search results based on user preferences or product relevance.
Index Optimization	Building and maintaining efficient indices to ensure fast retrieval of vectors.	Constructing tree or hash-based indices for large datasets, tuning for speed and accuracy.

Vector databases (in-memory databases) used to enable intelligent search, recommendations, and scalable real-time querying, making them critical for AI-driven applications

LangChain

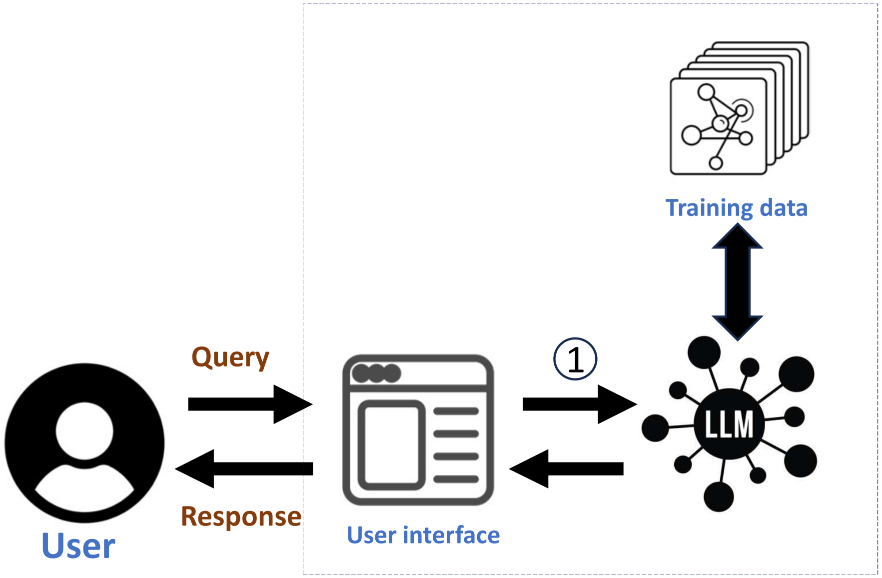
LangChain is an open-source framework designed to simplify the process of integrating LLMs with external data sources, tools, and workflows to create intelligent, context-aware applications like chatbots, retrieval-augmented generation (RAG) systems.

Core Components of LangChain

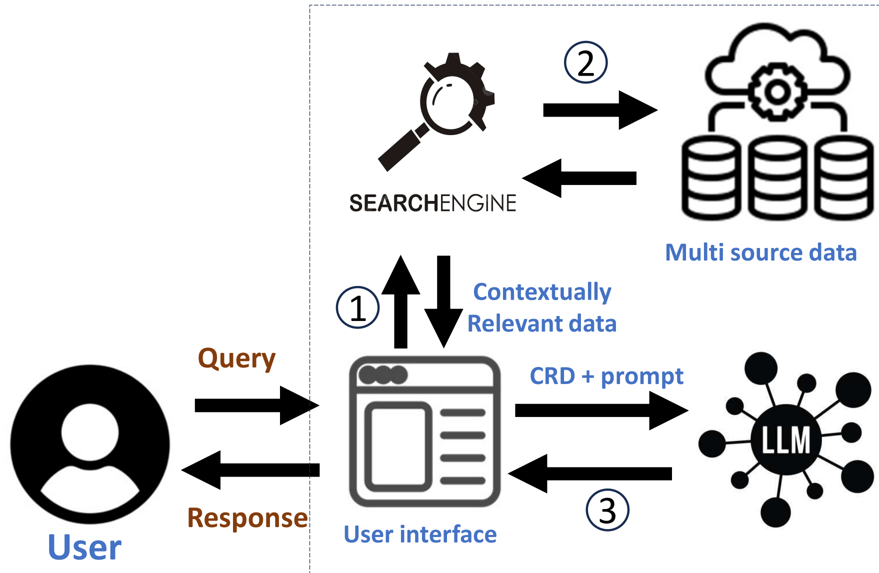
1. **LLM Wrapper**:
   - Interfaces for interacting with LLMs (OpenAI, Cohere, Hugging Face, etc.).
2. **Chains**:
   - Workflow logic that connects multiple steps, such as generating a prompt, calling an API, or retrieving data.
3. **Memory**:
   - State management to retain conversation context over multiple interactions.
4. **Agents**:
   - Autonomous decision-making modules that allow LLMs to choose and execute tasks dynamically.
5. **Tools and Connectors**:
   - Pre-built connectors for external tools (e.g., APIs, databases, Python functions).
6. **Retrievers**:
   - Interfaces to search and retrieve data from vector databases like Pinecone, Weaviate, and Milvus.
7. **Prompts**:
   - Utilities for creating and managing prompts, including template generation and formatting.

RAG

Low hallucination
Referencing system



Application using just LLMs



Application using RAG

- A model trained in 2023
- Cannot answer who won the world cup in 2024

- A model trained in 2023
- Can answer who won the world cup in 2024

The generative AI model training strategy consists of the ‘found content’ and the ‘question asked’ in the prompt

Datasets

- General purpose:
- https://commoncrawl.org/
- https://dumps.wikimedia.org/

Handson-1

- You are building a news aggregation app. You want to group the news articles based on the content into different categories. The app has a filter based on which you channel only news articles of specific category (selected by the user)

Handson-1

- You are building a news aggregation app. You want to group the news articles based on the content into different categories. The app has a filter based on which you channel only news articles of specific category (selected by the user)

Access news articles → Scrape data from web → Document clustering → Article listing and filtering → Articles rendering

Access news articles	Scrape data from web	Document clustering	Article listing and filtering	Articles rendering
NEWSAPI GDELT API ContextualWeb API Bing News API NYT API MediaStack API	beautifulsoup	Transformer + Classification + Summarization + Headline generation		Summarization + Link to the original source

Handson-2

- You are building an application to collect feedback from the customers in restaurants and provide a response after receiving a feedback.