Language Models

Fundamentals of Artificial Intelligence

Natural Language Processing (NLP)

field within AI and linguistics

aim: enable machines to understand and generate human language

language models: subfield of NLP

aim: model and understand language patterns (typically by learning to predict the next word in a sentence)

capabilities of large language models (LLM) cover most of NLP

NLP Tasks

- part-of-speech tagging
- named entity recognition
- text classification
- machine translation
- summarization
- question answering
- text generation (chatbots)
- code generation
- …

Symbolic vs Neural NLP

symbolic NLP = rules and logic

- uses hand-crafted rules, lexicons, and grammars
- relies on linguistic knowledge and logic-based systems

neural NLP = data and learning

- uses machine learning (especially deep learning)
- learns patterns from large datasets without explicit rules

Large Language Models (LLM)

recent hype around LLMs

fully started with ChatGPT release end of 2022

technical backbone: transformer architecture

transformers also applicable to computer vision (alternative to convolutional neural networks)

Tokenization

tokenization: *breaking text in chunks*

- word tokens: different forms, spellings, etc → undefined and vast vocabulary
- character tokens: not enough semantic content (longer sequences)
- popular compromise: byte-pair encoding

Embedding Layers

representation of entities by vectors

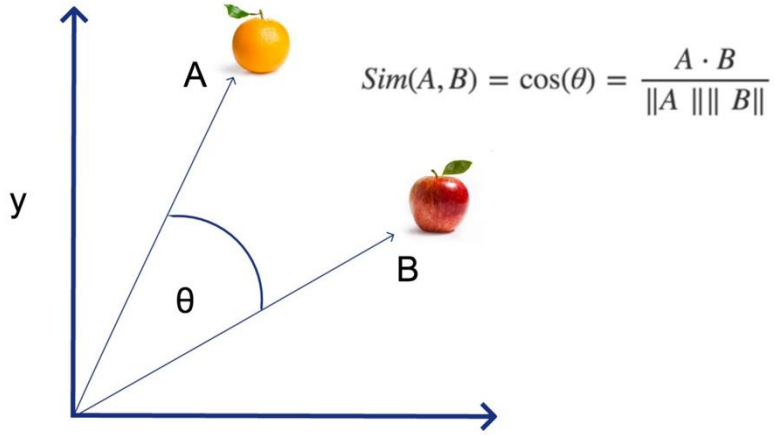
→ semantic similarity

most famous application: word embeddings

→ associations (natural language processing)

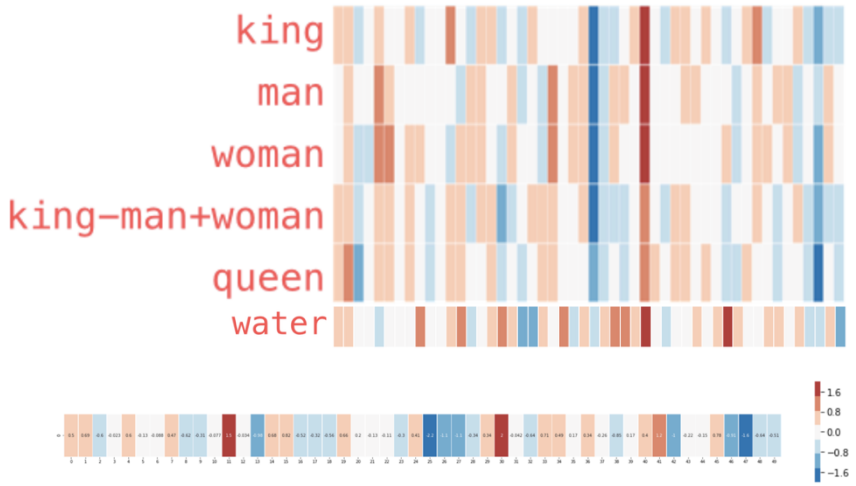
but general concept: embeddings of (categorical) features (e.g., products in recommendation engines)

learned via co-occurrence (e.g., word2vec)



but also direction of difference vectors interesting (analogies):

king − man + woman ~= queen



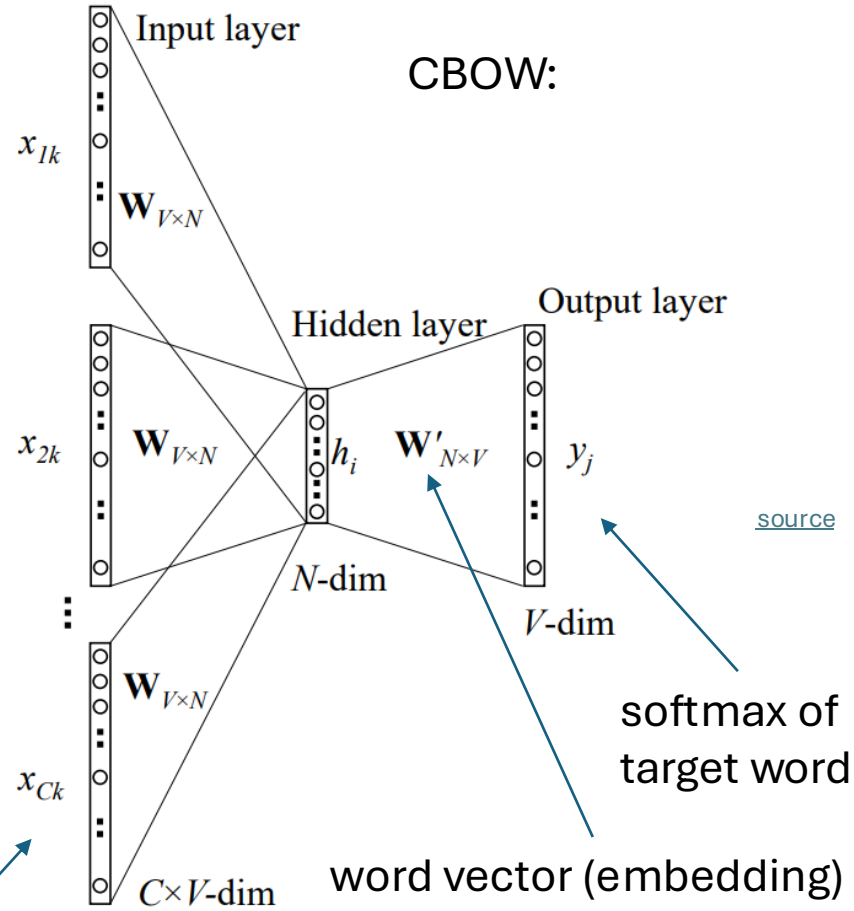
source

Some Thoughts on Word Embeddings

can be implemented as

- neural network with single hidden layer (linear activation)
- using, e.g., bag-of-words approach (predict masked word from its surroundings)

→ not context-aware



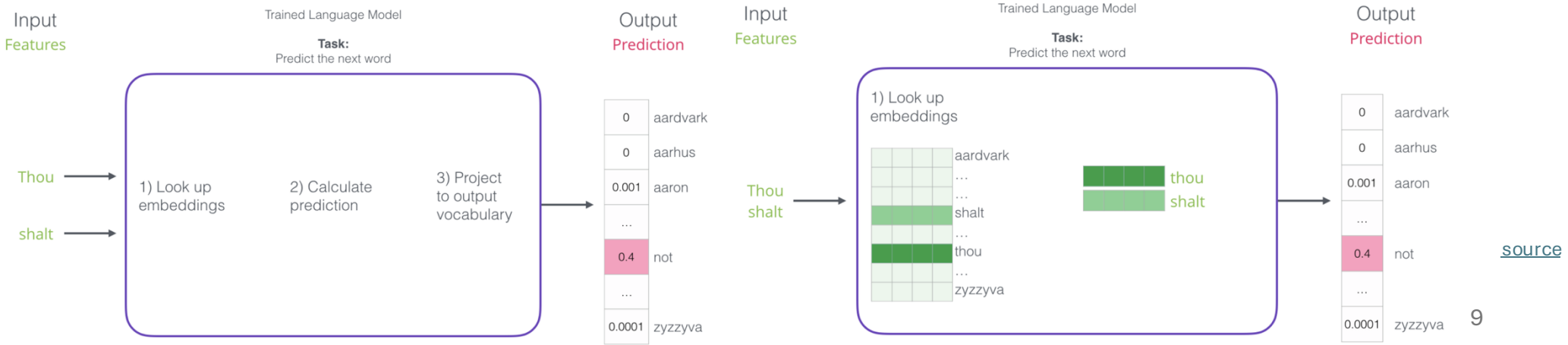
Word Embeddings as Part of Language Model

language models contain embedding matrix as part of learned parameters

typically several hundred dimensions for word vectors (to be compared with vocabulary sizes of many thousands)

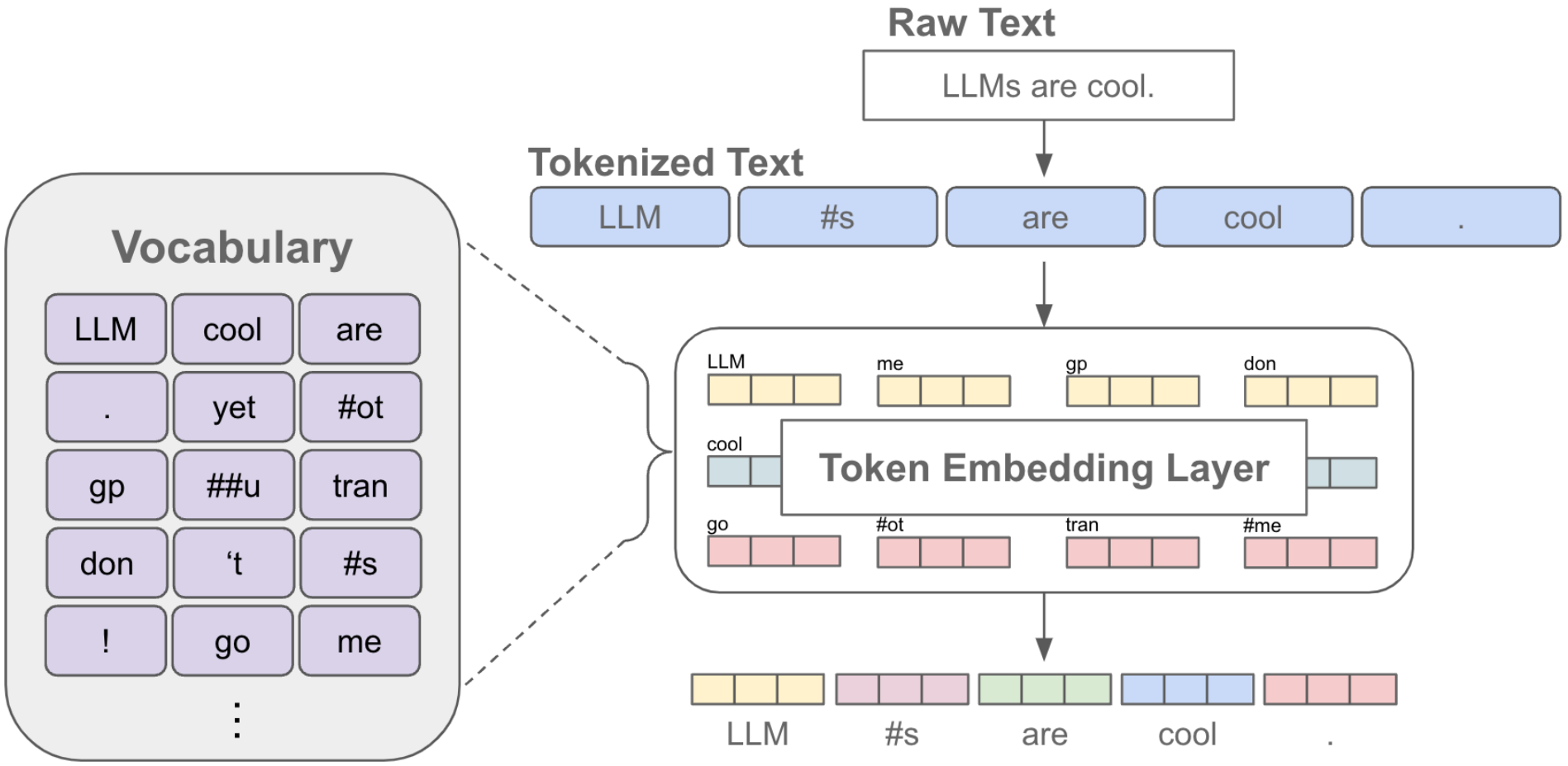
(can also be extracted and subsequently used as pretrained embeddings for other task)

next-word prediction:



source

Tokenization & Embeddings



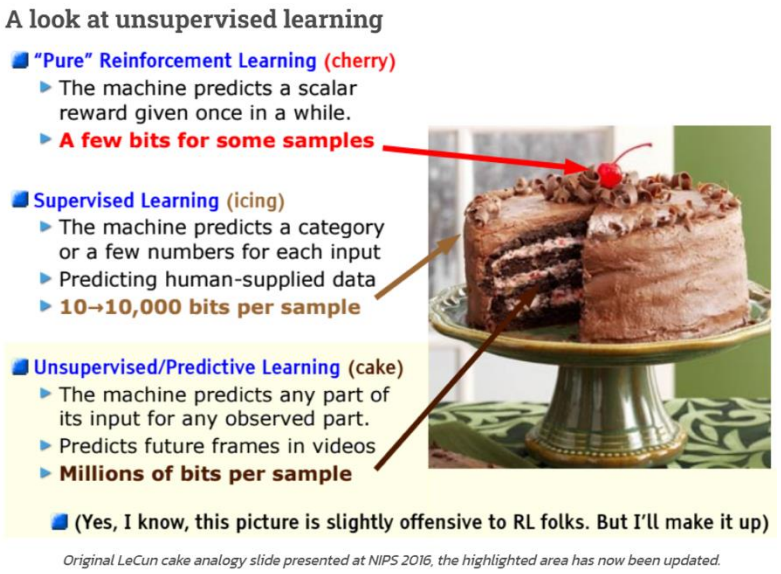
source

Self-Supervised Learning

ML needs lots of training data

unsupervised learning (learning by observation)

no target information → kind of "vague" pattern recognition (but plenty of data)



source

can be cast as **self-supervised learning**:

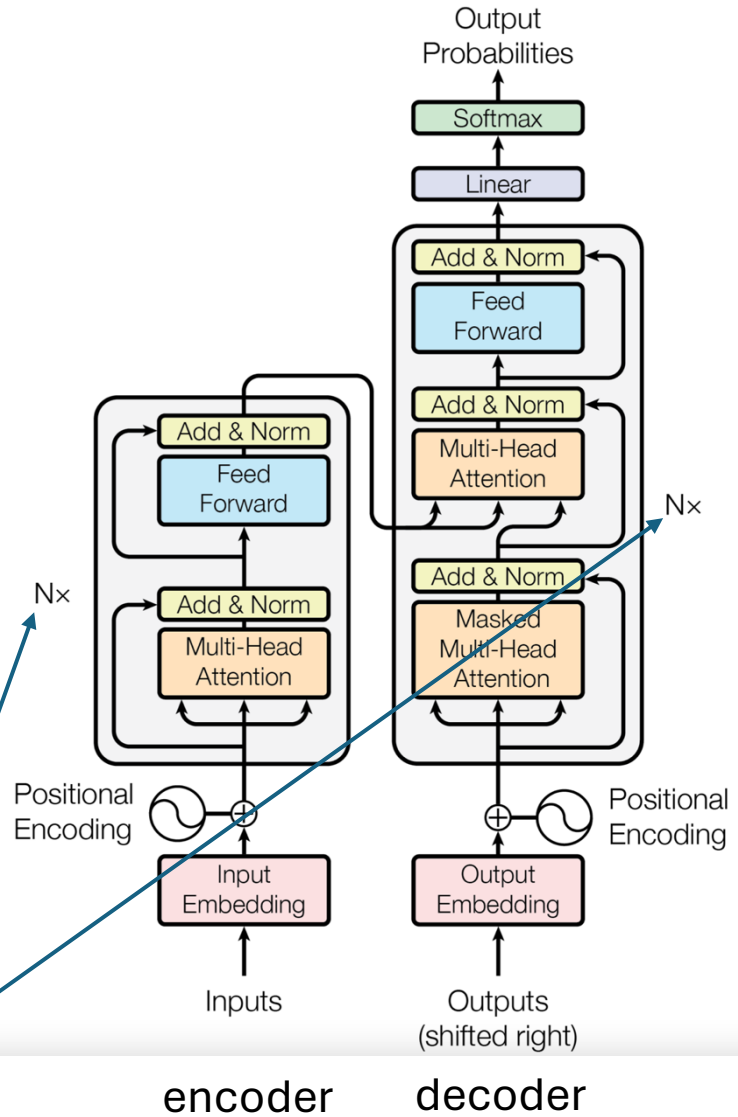
- input-output mapping like supervised learning
- but generating labels itself from inputs

```
The Lord of the Rings is considered one of the greatest
The Lord of the Rings is considered one of the greatest fantasy
                        ⋮
```

now run this across the entire internet …

Context Awareness: Transformer

original transformer:
sequence-to-sequence model
(e.g., for machine translation)



[Attention Is All You Need]:

completely replaced recurrent neural networks (RNN)

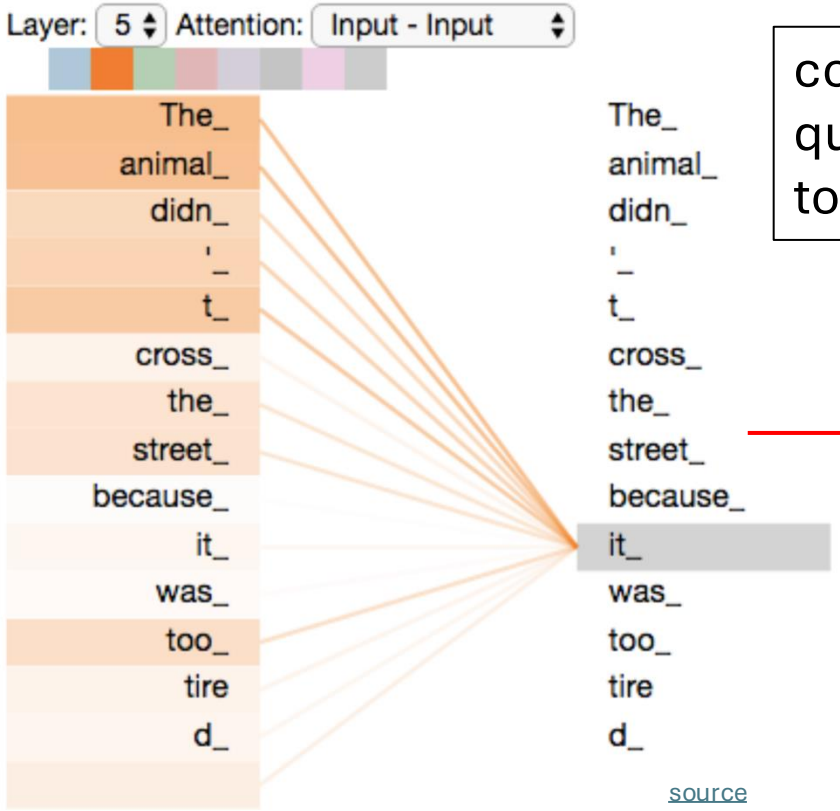
much more parallelization → bigger models

better long-range dependencies thanks to shorter path lengths in network (less sequential operations)

encoder/decoder stacks (depth):
output fed as input to next one

Self-Attention Mechanism

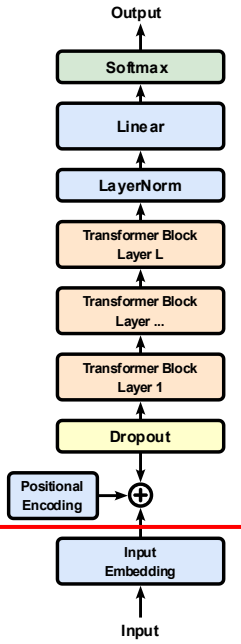
evaluating other input tokens in terms of relevance for encoding of given token

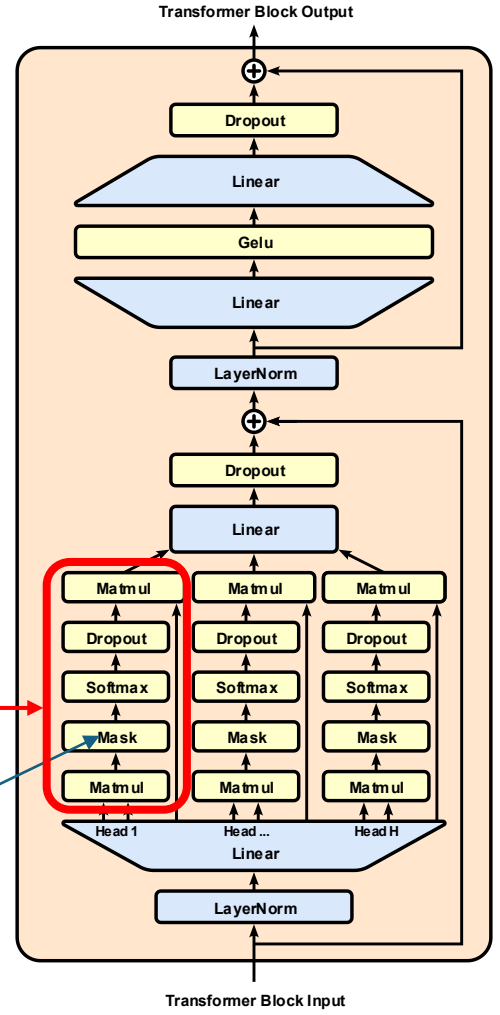


source

computational complexity quadratic in length of input (each token attends to each other token)

GPT (decoder-only):





only allow to attend to earlier positions in sequence (masking future positions)

Scaled Dot-Product Attention

softmax not scale invariant (largest component dominates for large scale) and dot product larger for more vector dimensions

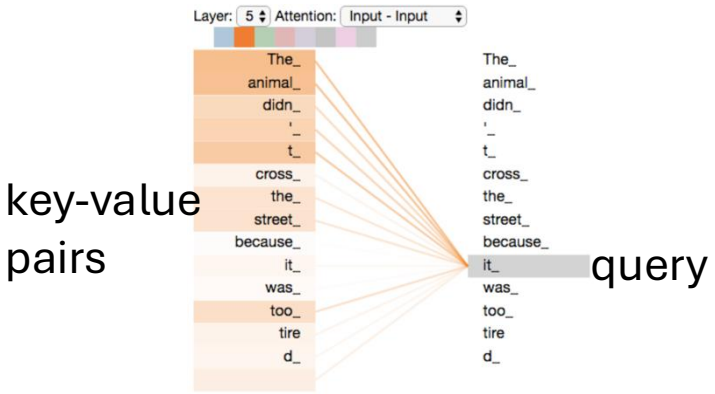
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention

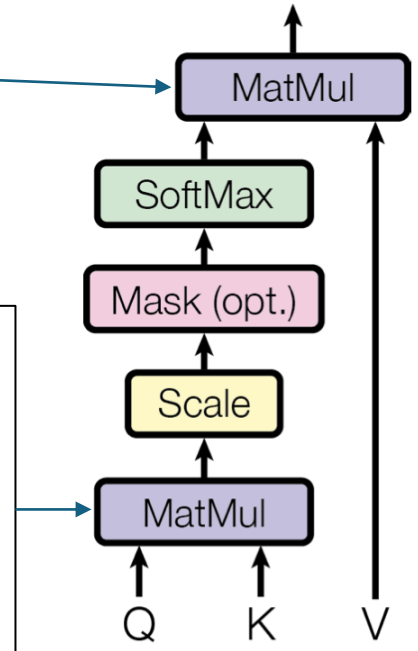
3 matrices created from input embeddings by multiplication with 3 different weight matrices

- query Q
- key K
- value V

filtering: multiplication of attention probabilities with corresponding key-word values



scoring each of the key words (context) with respect to current query word: correlation between inputs (not independent as in conventional neural networks)



source

Example: Dimensions in BERT

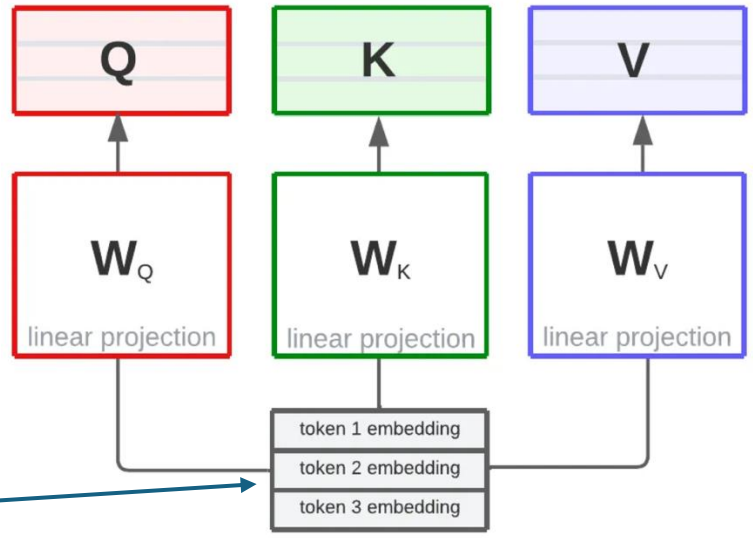
$$W_Q, W_K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

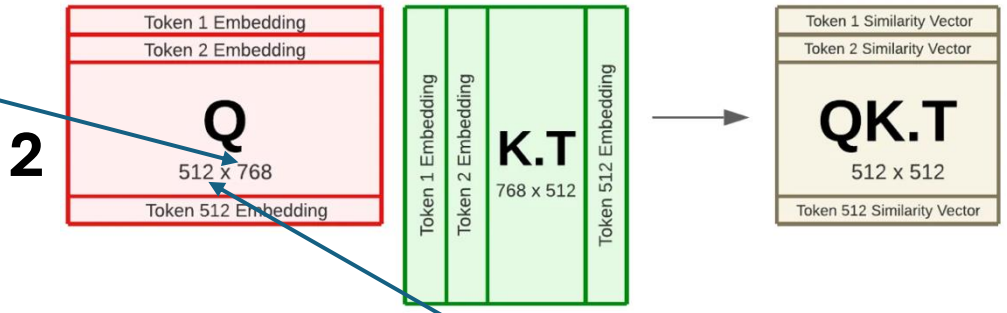
often $d_k = d_v = d_{\text{model}}$
(but not necessarily)

linear projections

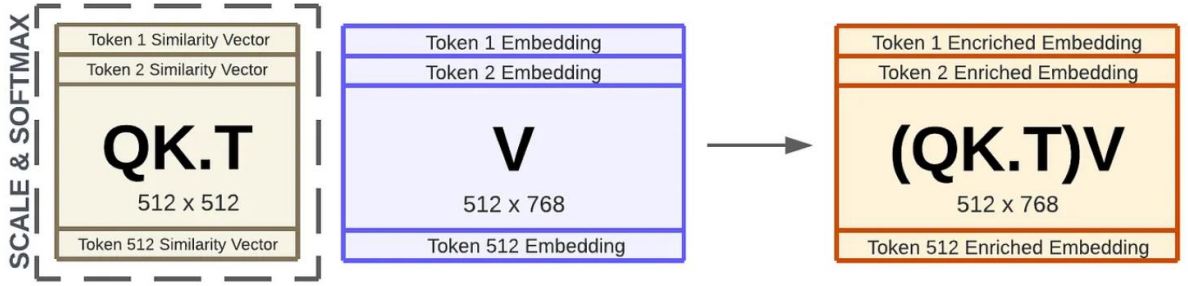
learned parameters

inputs



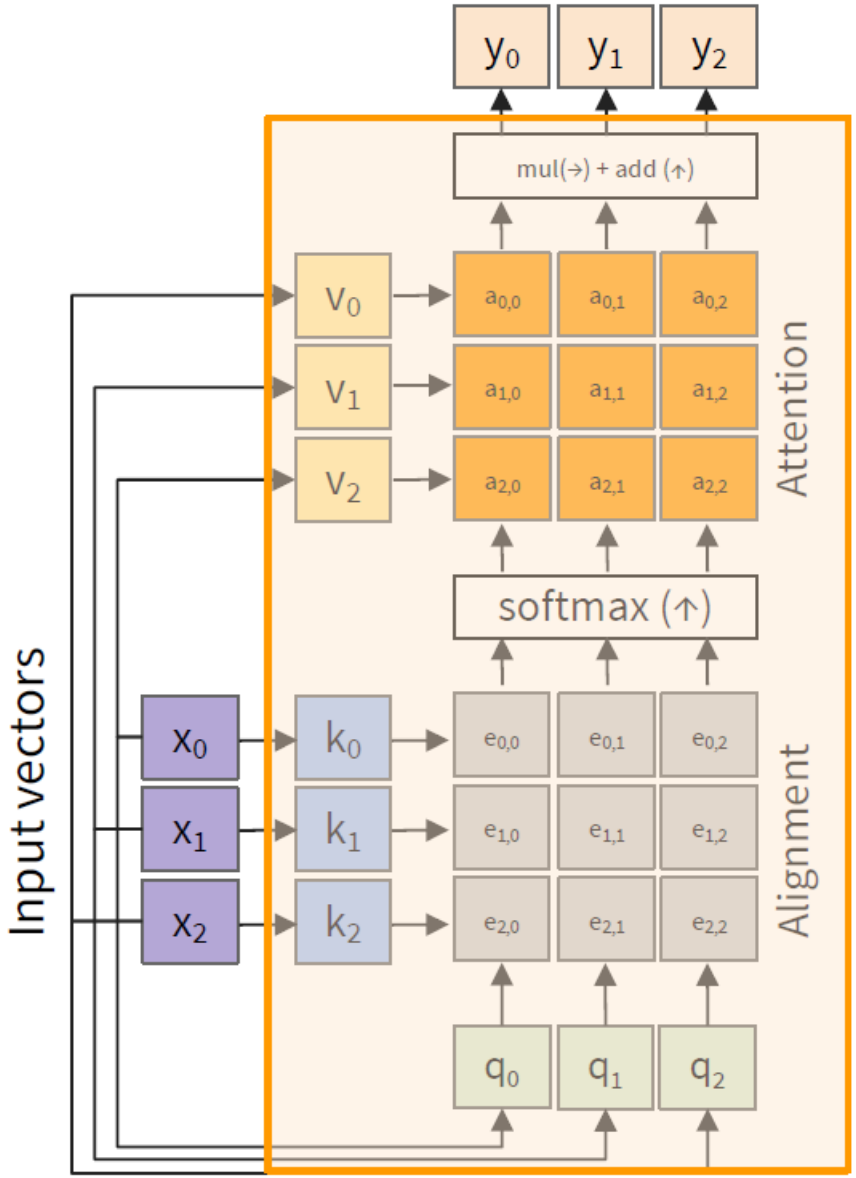


sequence length n
upper bound: context length of model
(design choice)



source

MatMul



Outputs:
context vectors: y (shape: D_v)

d_v

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

d_k

d_{model}: embedding size

Inputs:
Input vectors: x (shape: N x D)

weighted average: reflecting to what degree a token is paying attention to the other tokens in the sequence

Attention Mask

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V$$

attention mask

causal masking:

dynamically mask future positions in sequence by setting to $-\infty$

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Padding Mask

goal: (technically) batch together sequences of different lengths

→ include fake tokens to pad shorter ones to the max length in the batch

then add a padding mask to let the model ignore these fake tokens:

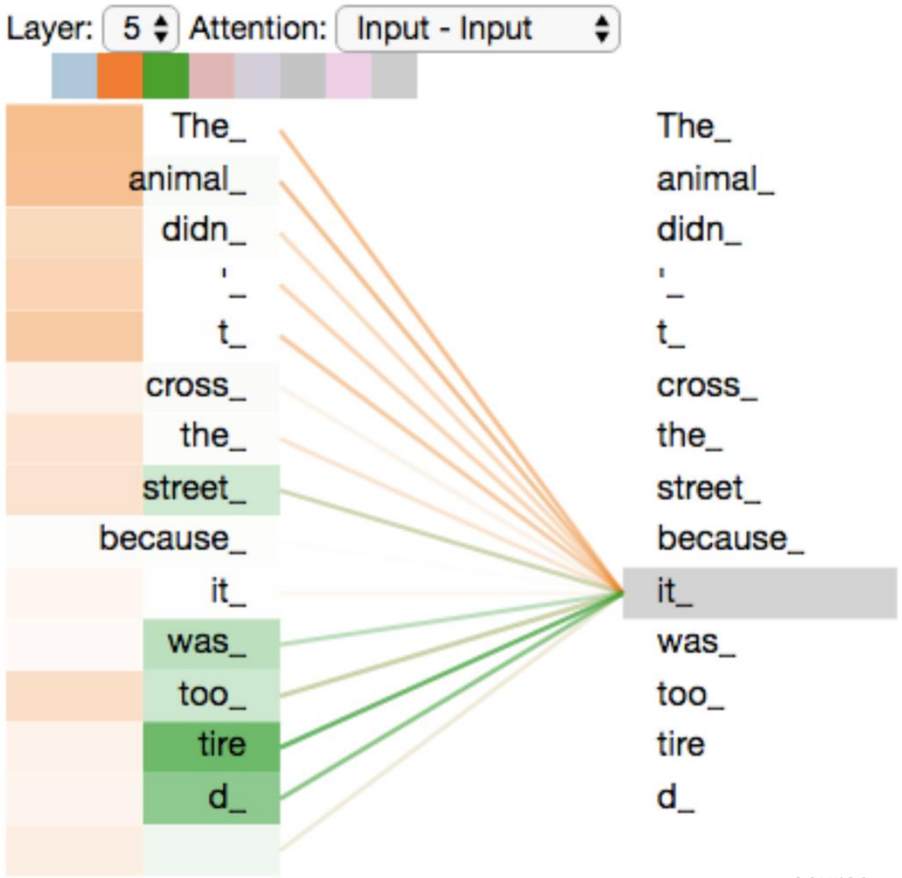
$$M = M_{\text{causal}} + M_{\text{padding}}$$

$$M_{\text{padding}}(i, j) = \begin{cases} 0 & \text{if token } j \text{ is valid} \\ -\infty & \text{if token } j \text{ is padding} \end{cases}$$

Multi-Head Attention

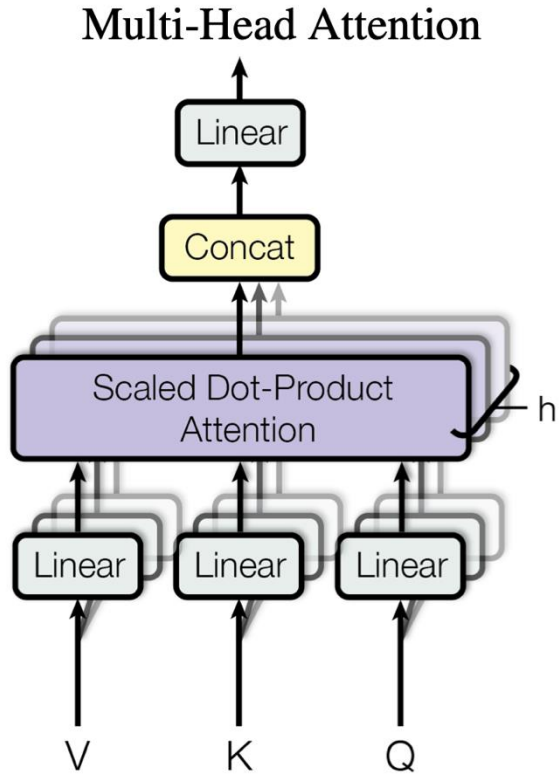
often $d_k = d_v = d_{\text{model}}/h$
and all h heads share the same d_k and d_v

multiple heads: several attention layers running in parallel



source

different heads can pay attention to different aspects of input (multiple representation sub-spaces)



source

Positional Encoding

defines context length

attention permutation invariant → need for positional encoding to learn from order of sequence

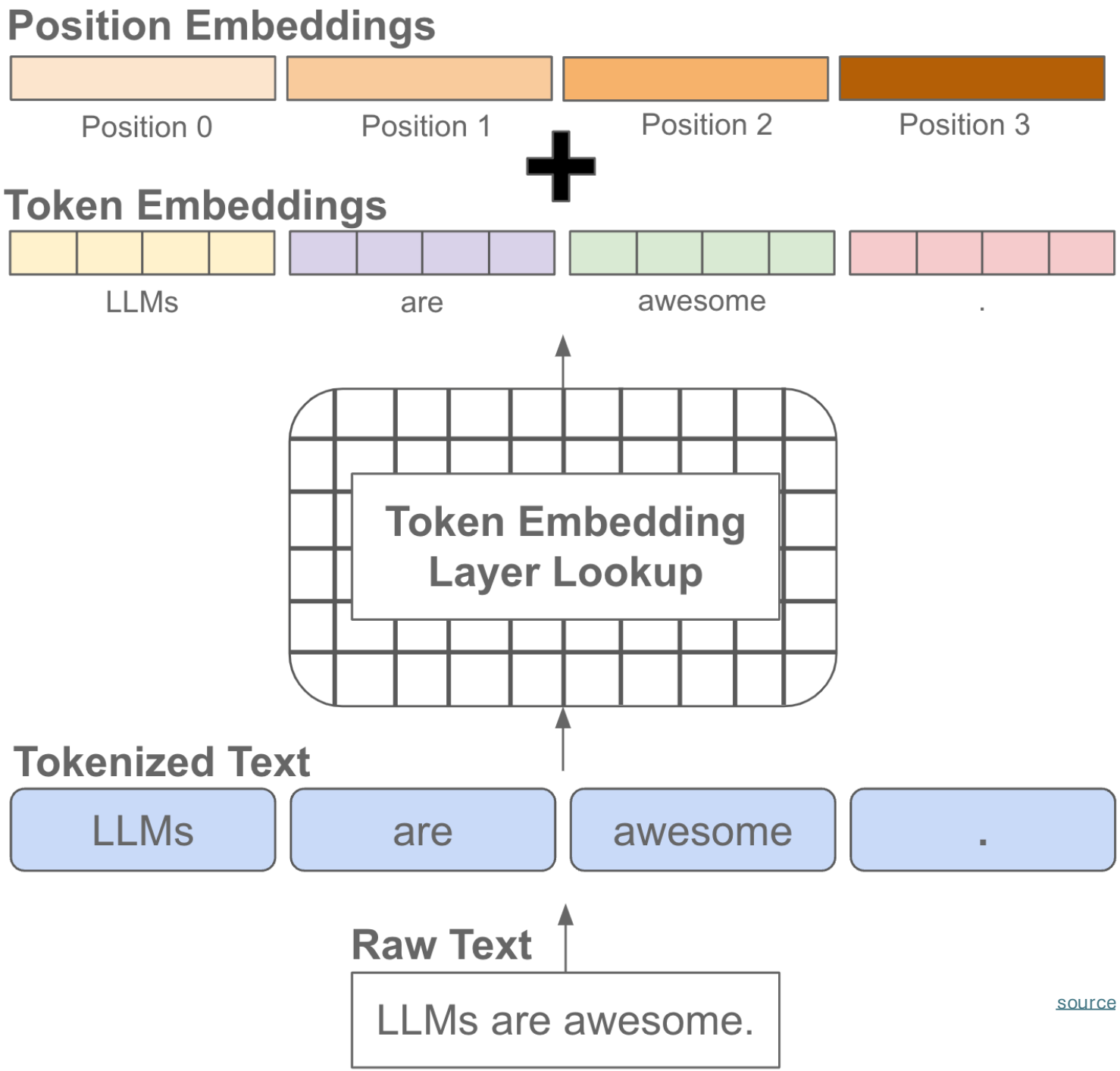
different options:

- for each absolute position (from 1 to maximum sequence length), add a learned vector (of dimension d_{model}) to input embeddings → each positional embedding independent of others
- add fixed sinusoidal functions for each position and dimension i

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \qquad PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

- rotate input embeddings by multiples (position) of a small angle (RoPE) → captures both relative and absolute positions

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$


Position Embeddings
Position 0
Position 1
Position 2
Position 3
Token Embeddings
LLMs
are
awesome
.
Token Embedding Layer Lookup
Tokenized Text
LLMs
are
awesome
.
Raw Text
LLMs are awesome.

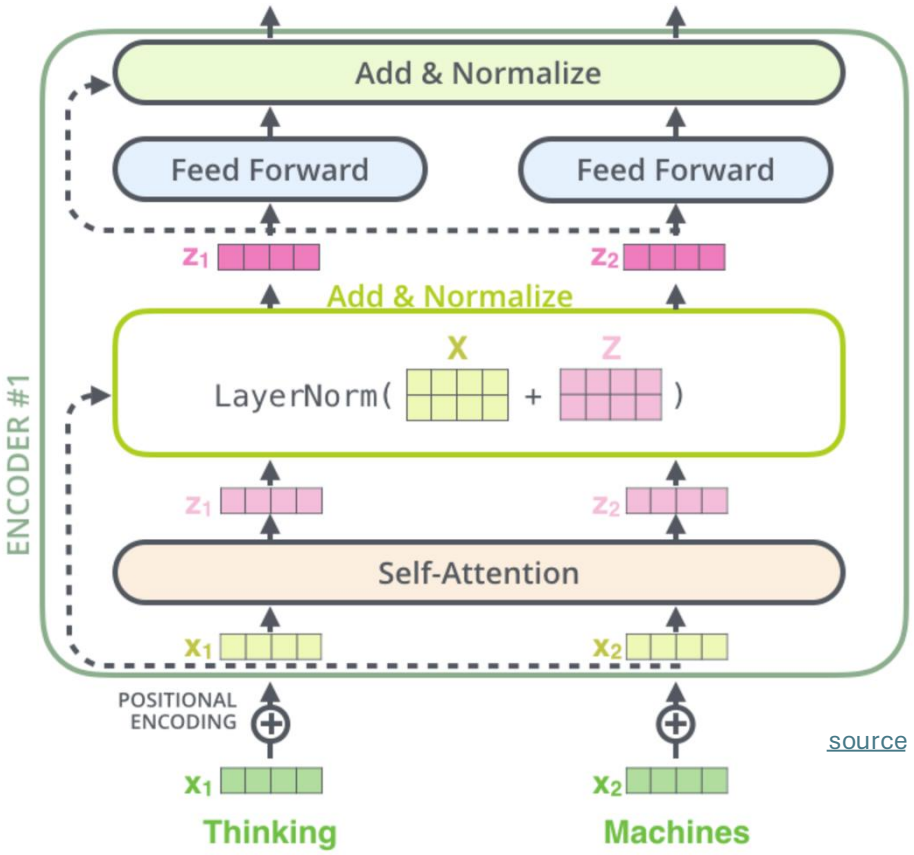
source

Skip Connections and Layer Normalization

skip connections improve robustness by

- preserving original input (attention layers as filters) as well as gradients (mitigate vanishing-gradient problem)
- easier learning of identity functions (useful for disregarding modules that do not improve model performance)

layer normalization improves stability, convergence, and generalization by normalizing activations per input token

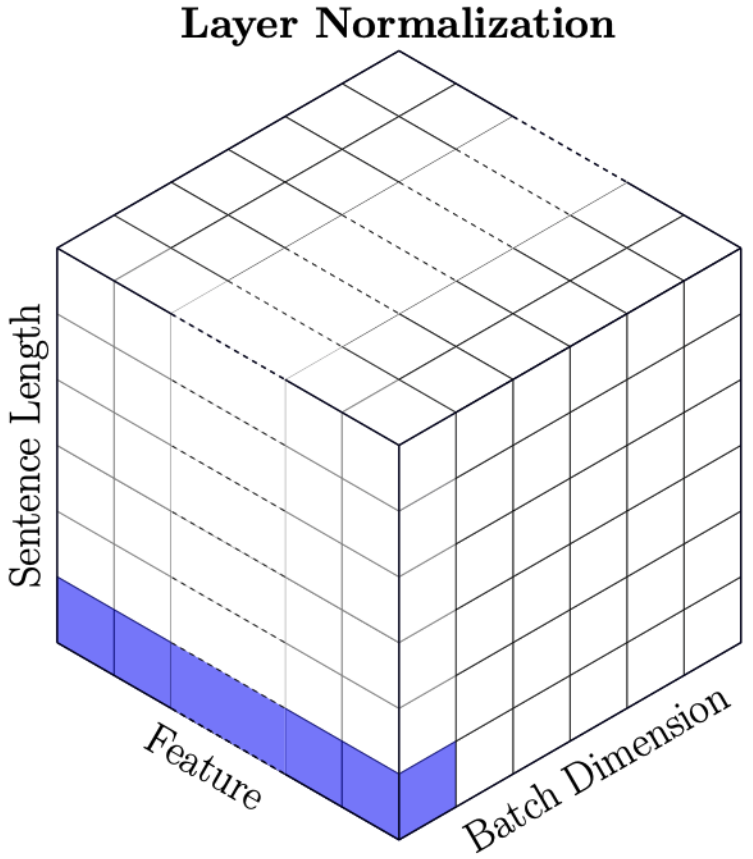


source

Batch vs Layer Norm

batch norm typically in computer vision

layer norm in NLP (variable-sized inputs)

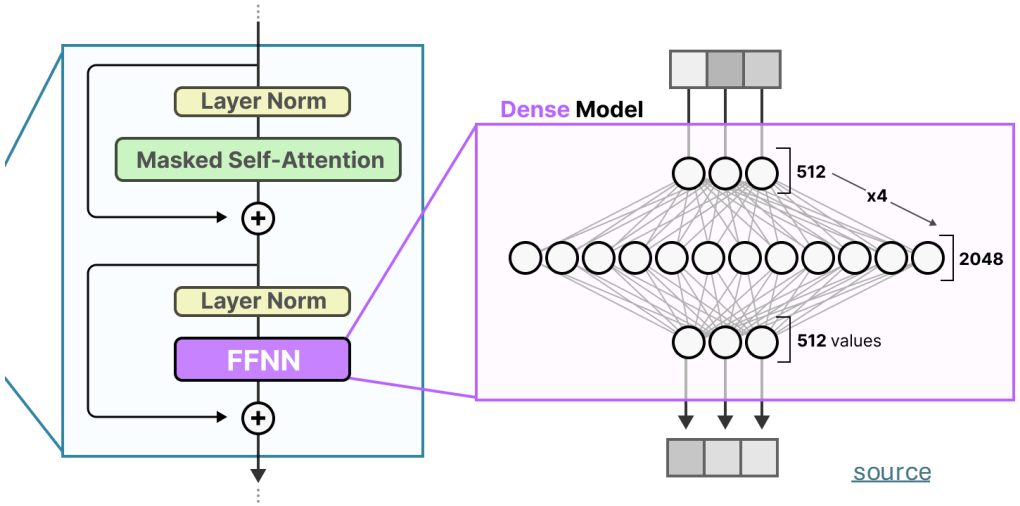


Role of Feed-Forward Neural Networks (FFNN)

position-wise FFNNs:

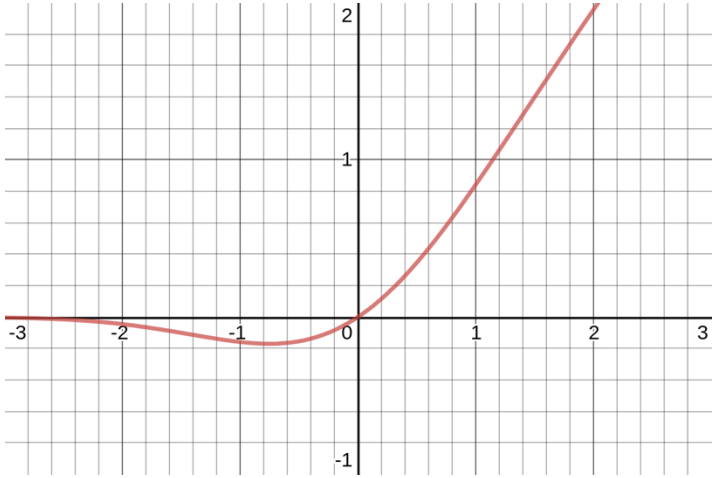
following each attention layer, apply an identical FFNN (with two layers) independently to each embedding vector

(correspond to bulk of overall parameters)



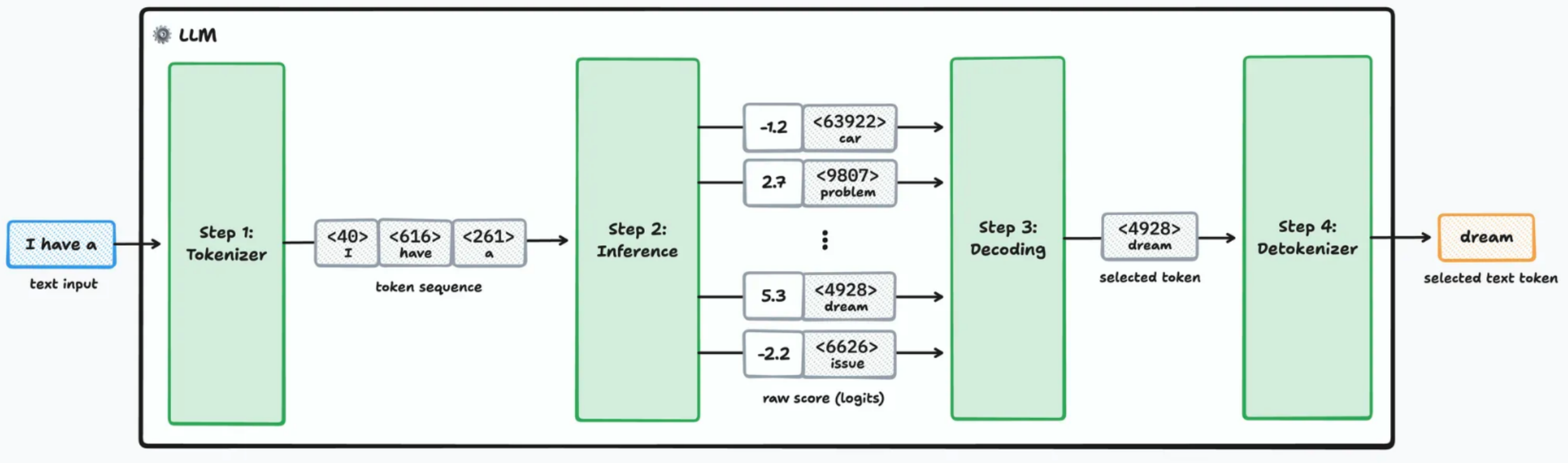
can be interpreted as compute after connect (attention)

attention is just weighted averaging
→ need for non-linearities (often GeLU activations) to capture more complex patterns



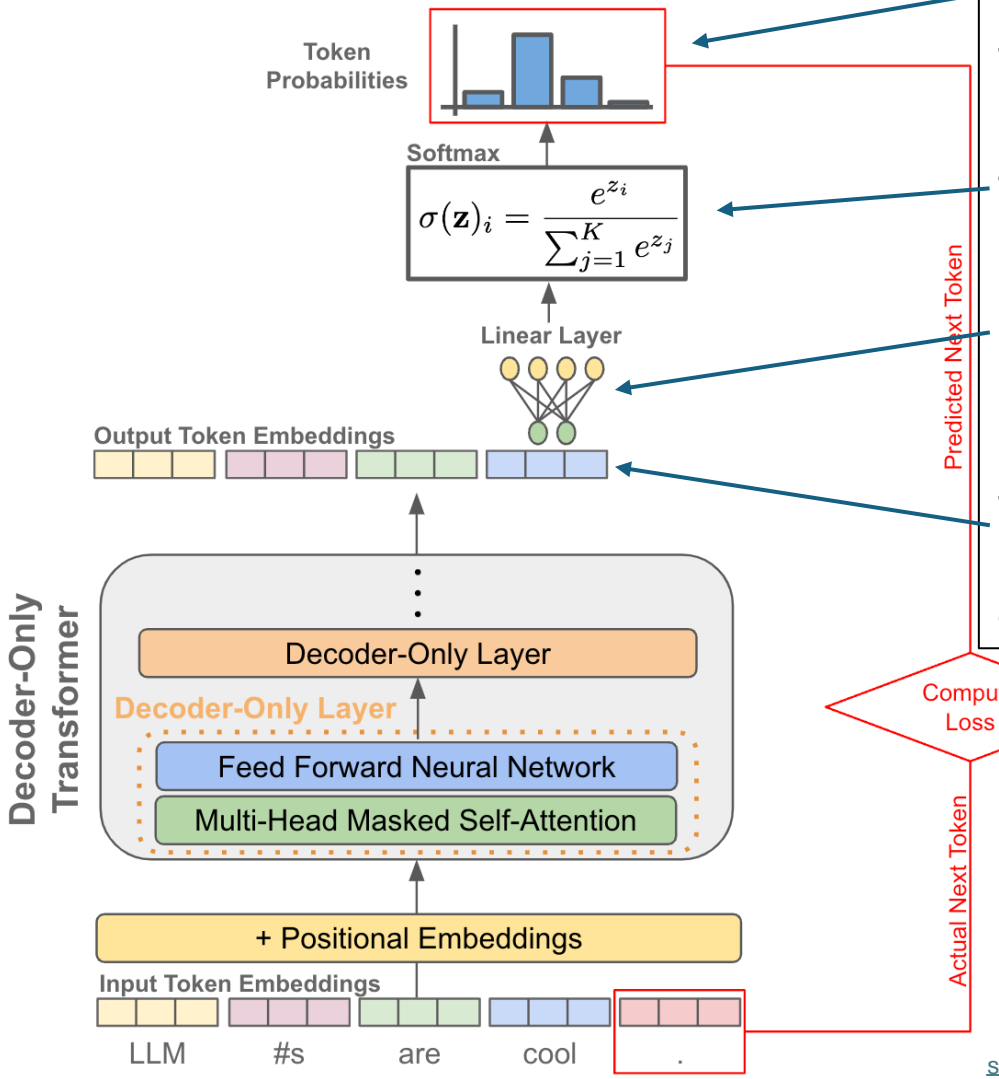
Next-Token Prediction

forward pass:



source

From Embeddings to Probabilities



source

next-token probabilities for each token in vocabulary

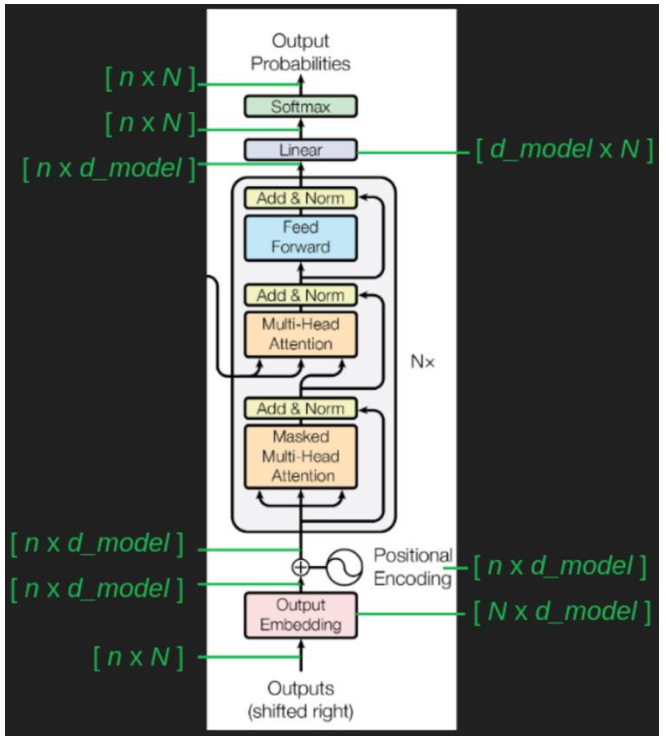
transformation to probabilities

de-embedding to vocabulary: projection to vector of logits

for projection, only use contextual embedding of last token (still need all contextual embeddings due to matrix multiplication in self-attention)

(masked token for encoder-only transformers)

n: sequence length
N: vocabulary size
d_model: embedding size



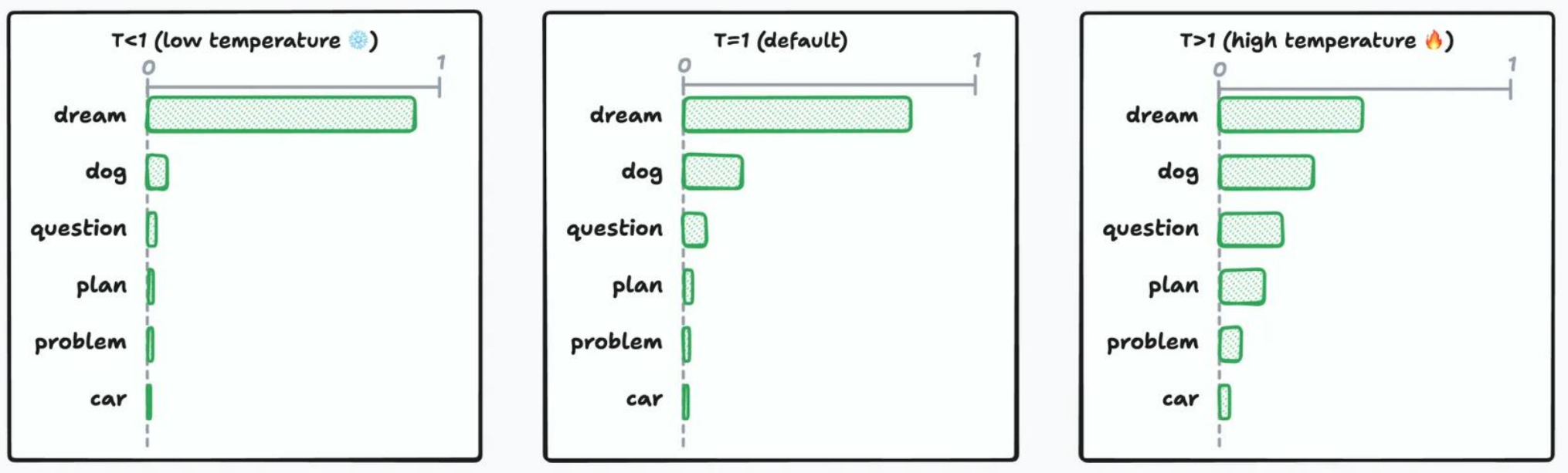
source

Token Selection at Inference

typically, pick according to probabilities

degree of randomness can be controlled by temperature parameter T

$$\text{softmax}(z_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{n} e^{z_j/T}}$$

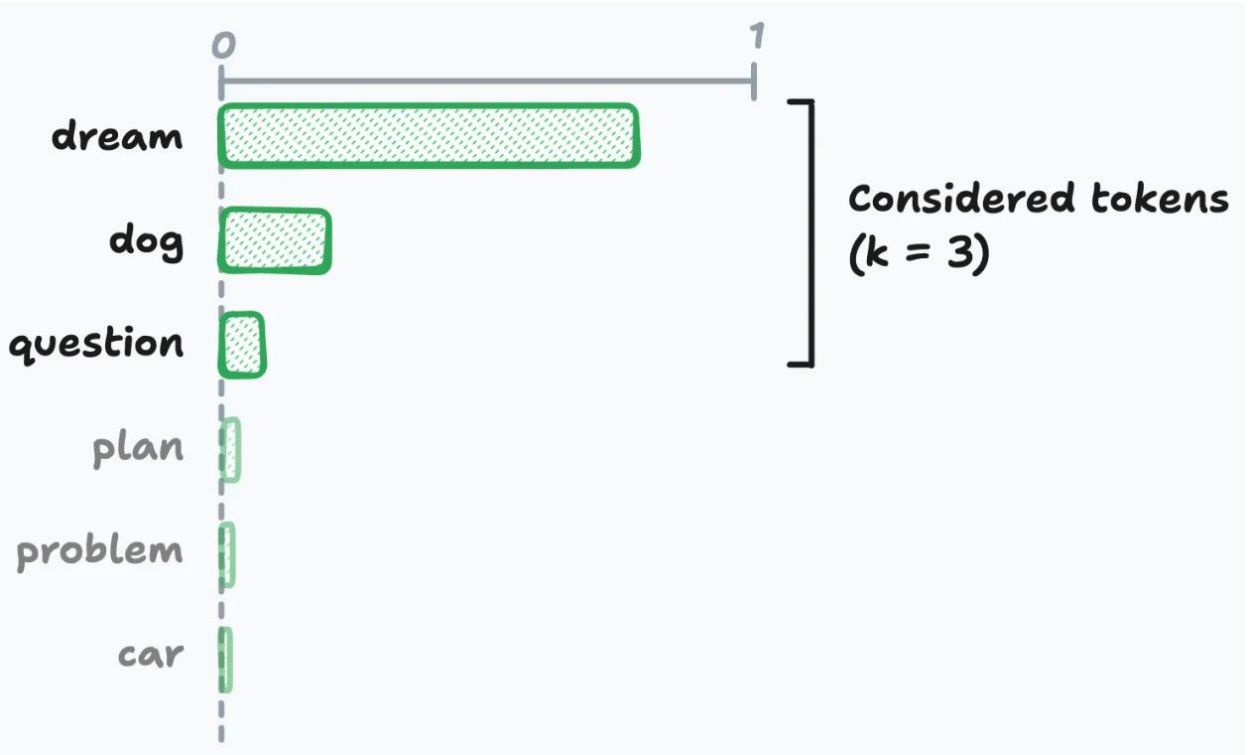


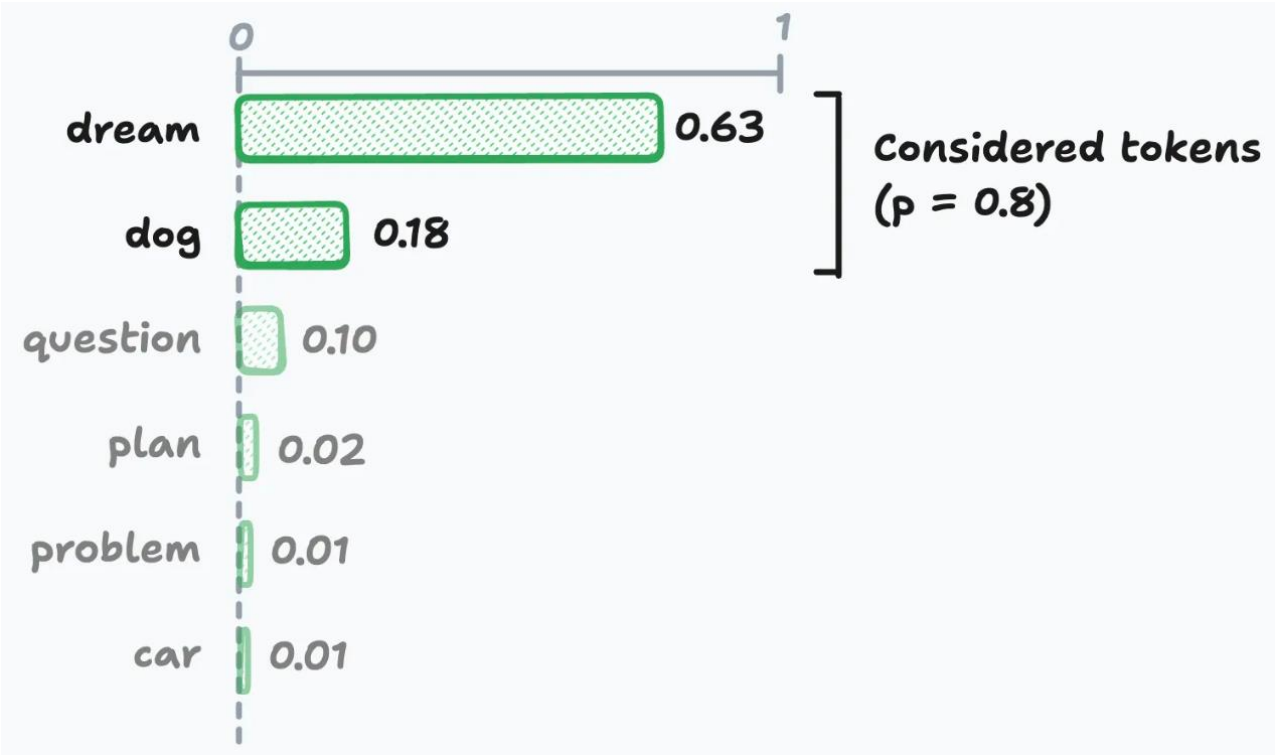
source

$T \to 0$ → greedy

creativity ;)

top-k & top-p Sampling



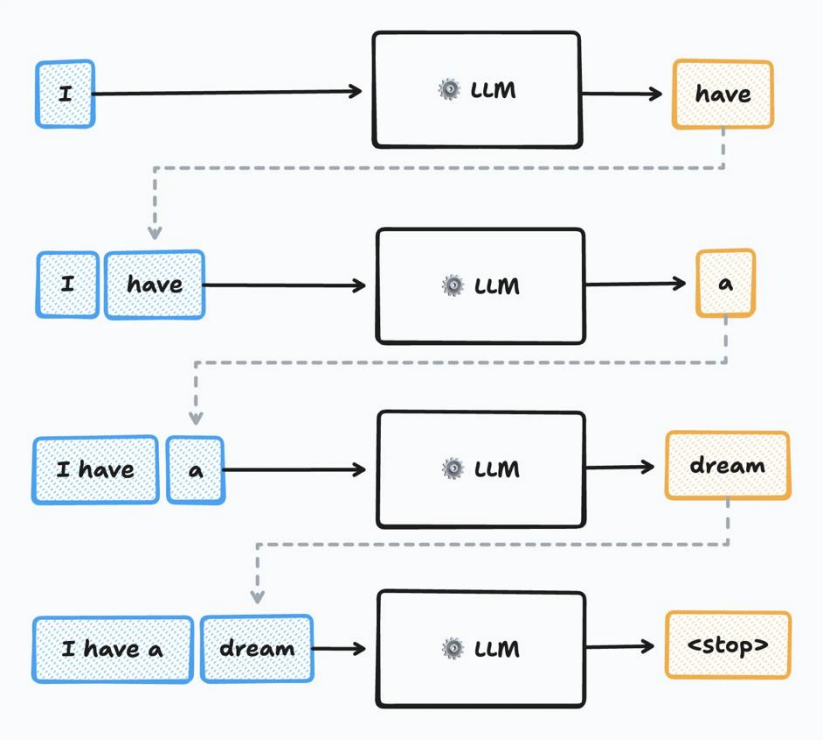


source

Sequence Completion

autoregressive:

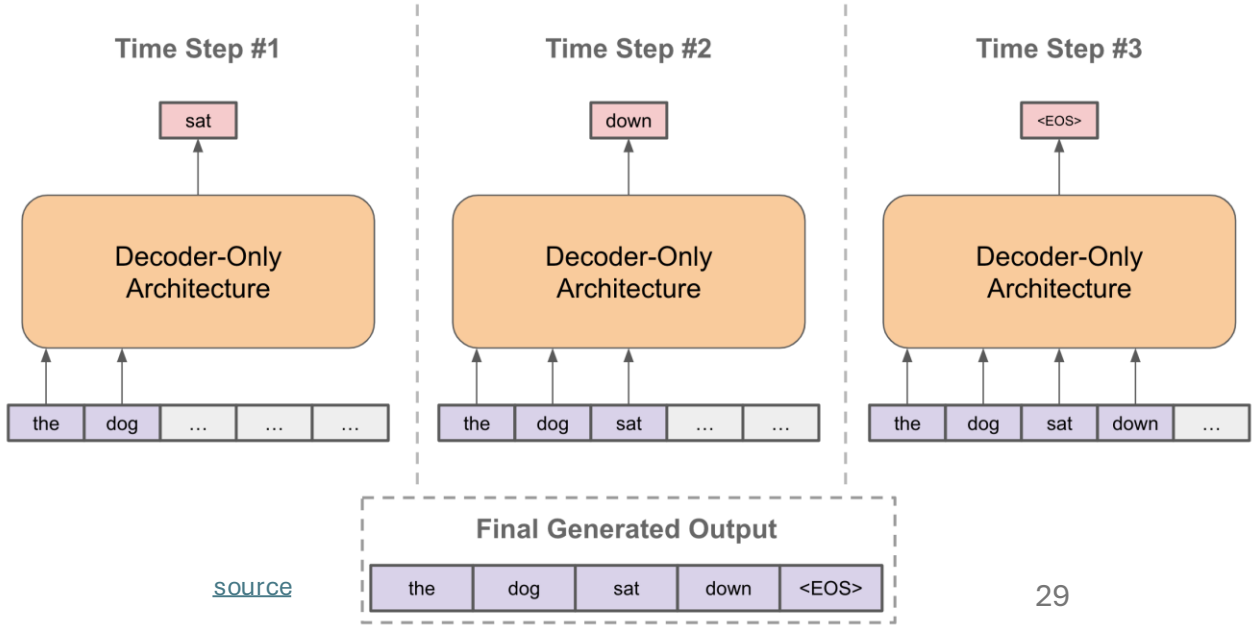
at each step, choose one output token, then add it to decoder input sequence for next step



source

prompt:

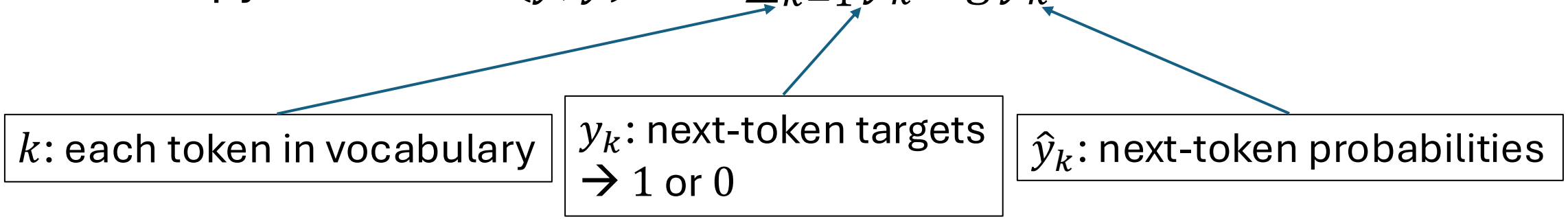
externally given initial sequence for running start and context on which to build rest of sequence



source

Training Loss

cross-entropy loss: $L(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$



during training, a next-token-prediction loss is calculated for every position in input sequence (not just the last one)
→ earlier positions have shorter effective contexts (masking of future positions)

reason: mirroring actual inference case of autoregressive text generation and arbitrary prompts

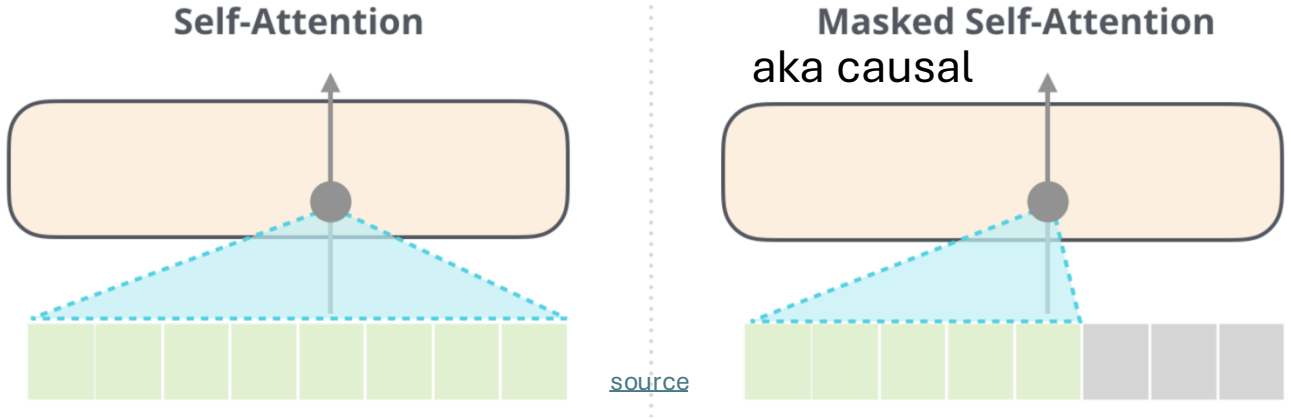
total loss = sum of individual losses

Transformer Types

encoder-only transformers:

- goal: language understanding
- training: masked-token prediction

prime example:
BERT



source

decoder-only transformers:

- goal: text generation
- training: next-token prediction

prime example:
GPT

encoder-decoder transformers:

- goal: sequence-to-sequence models
- training: next-token prediction

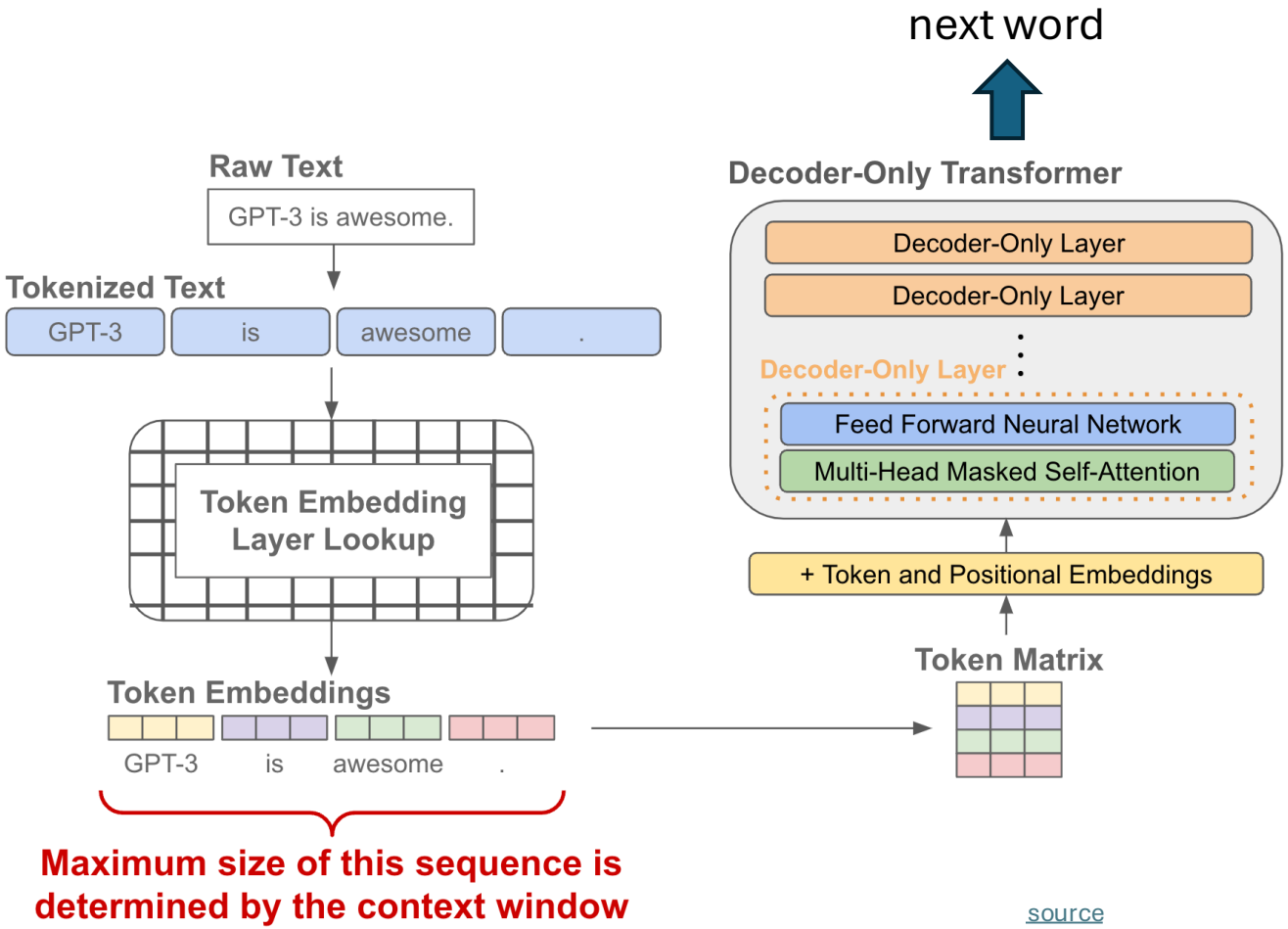
Decoder-Only = Autoregressive Models

today typically synonymous with LLM

stack of transformer decoders

→autoregressively generated sequence of tokens (directly usable as backbone of a chatbot)

training objective: next-token prediction from context



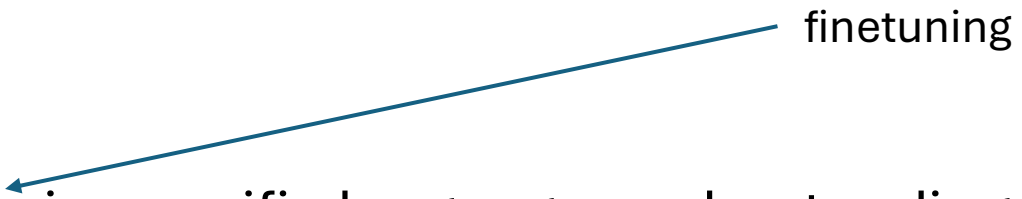
source

Encoder-Only = Bidirectional Models

training objective: masked tokens to be predicted from context

→ aka masked language models (MLM)

bidirectional: jointly conditioning on both left and right context



stack of transformer encoders

→ contextual embeddings (for later use in specific language-understanding tasks like text classification, sentiment analysis, named entity recognition)

still needs a decoder head for training purposes, using the hidden state of the masked token (dedicated [MASK] token) for projection to vector of logits

Autoregressive Text Generation

LLMs (autoregressive transformers) generate one output token at a time (according to predicted probabilities)

then add it to the context for the prediction of the next token

→ resulting in a full text, sampled from predicted probabilities

How are LLMs Generative Models then?

at each step, take the sequence of previous tokens $x_{<t}$ (context) and predict a conditional probability distribution over the next token x_t:

$$P(x_t|x_{<t})$$

joint probability distribution over text sequences via chain rule of probability:

$$P(x_1, x_2, \cdots, x_T) = \prod_{t=1}^{T} P(x_t|x_{<t})$$

→ no direct prediction of joint probability in one shot, but implicit calculation through product of conditionals

Prompting

feed information into decoder-only transformer via input prompt

→ attention to context

using all the knowledge acquired during internet-scale pretraining and stored in weights

kind of programmable neural network

→ a new paradigm: **one model for many tasks**

In-Context Learning (ICL)

the model's ability to condition its outputs on patterns, instructions, or demonstrations present in the context window, *without updating parameters* (in contrast to fine-tuning)

→ not really learning (rather inference-time pattern recognition: locating and using relevant learnings from training)

ICL is the emergent phenomenon that makes prompting powerful.

LLMs

autoregressive text generation of decoder-only transformers allows prompting of models (can output text to arbitrary topics)

→ enables chatbots like ChatGPT

triggered hype about generative AI and started an LLM scaling race (the larger the model, the better the capabilities)

lightweight PyTorch re-implementation of GPT: minGPT
more powerful: nanoGPT, LitGPT

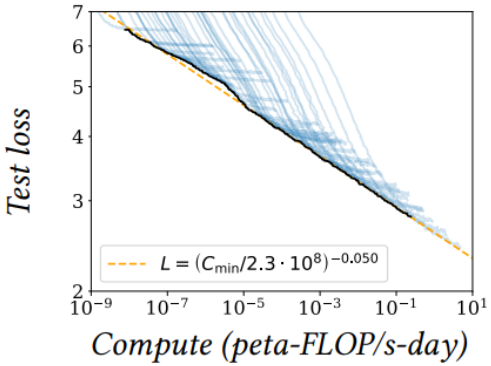
Size Matters: **LARGE** Language Models

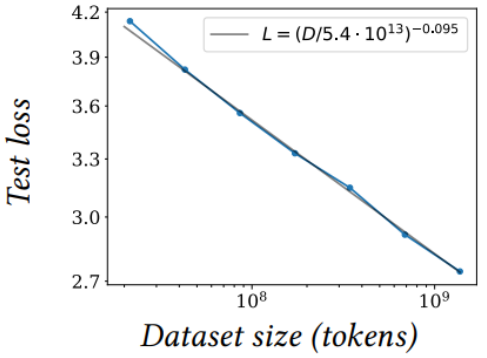
scaling laws, Chinchilla: coupled performance power laws with model size, amount of training data, and compute

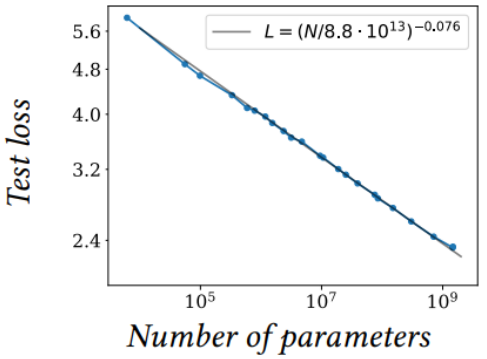
→ era of large-scale models

emergent abilities of LLMs:

- multi-task learning: perform new tasks at test time without task-specific training (via prompting)
- reasoning capabilities







source

Some LLM Numbers

example Kimi K2 (Mixture-of-Experts model):

as usual, comes in base and instruct variants

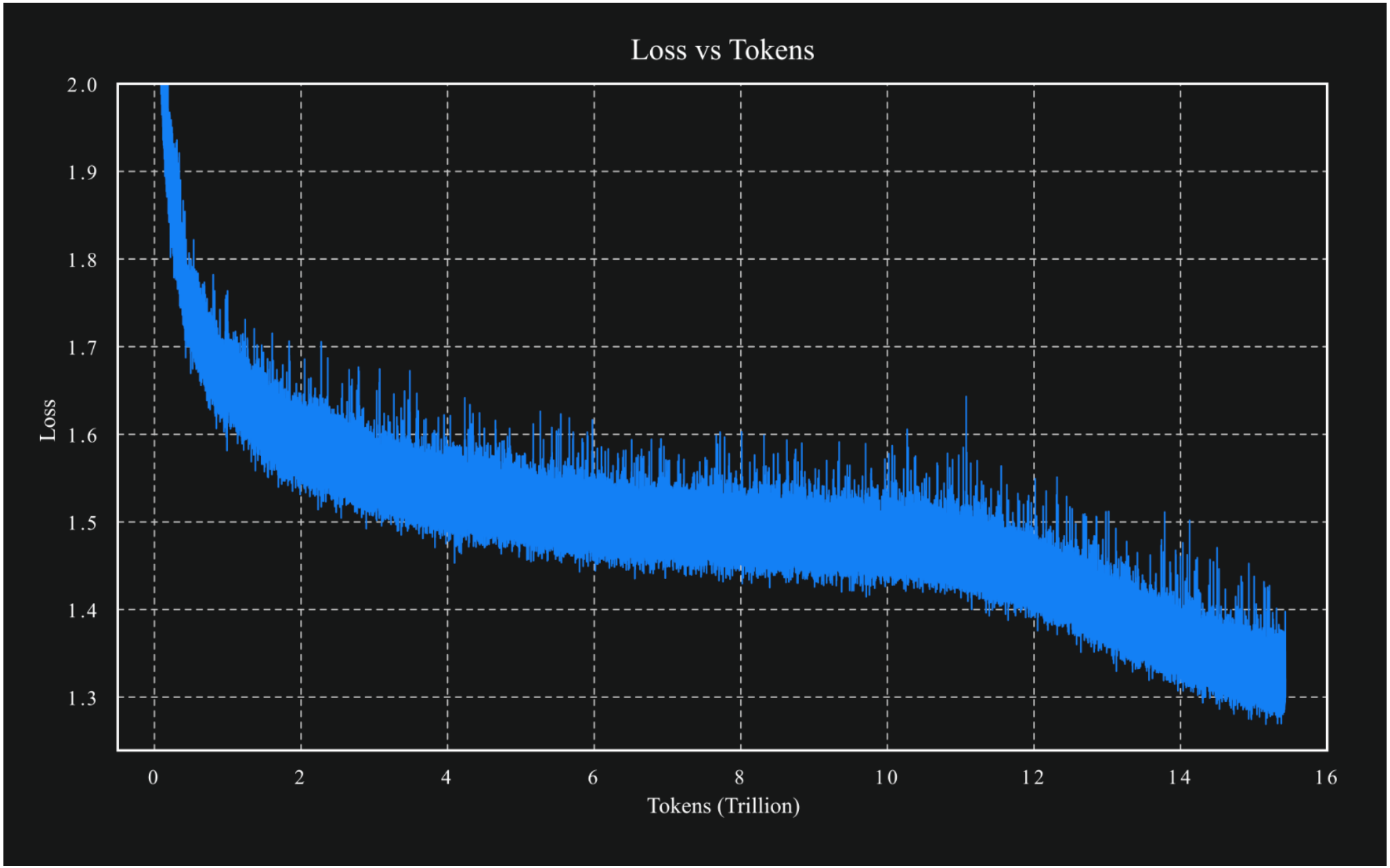
- vocabulary size (tokens): 160K
- embedding dimension: 7,168
- context length (tokens): 128K

- total parameters: 1T
- training tokens: 15.5T

→ a lot of memorizing

compression of the internet

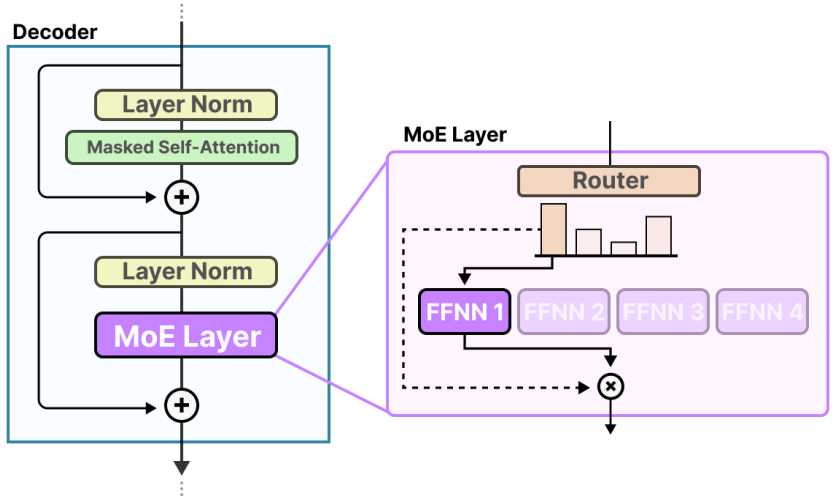
- number of layers: 61
- number of attention heads: 64
- number of experts: 384

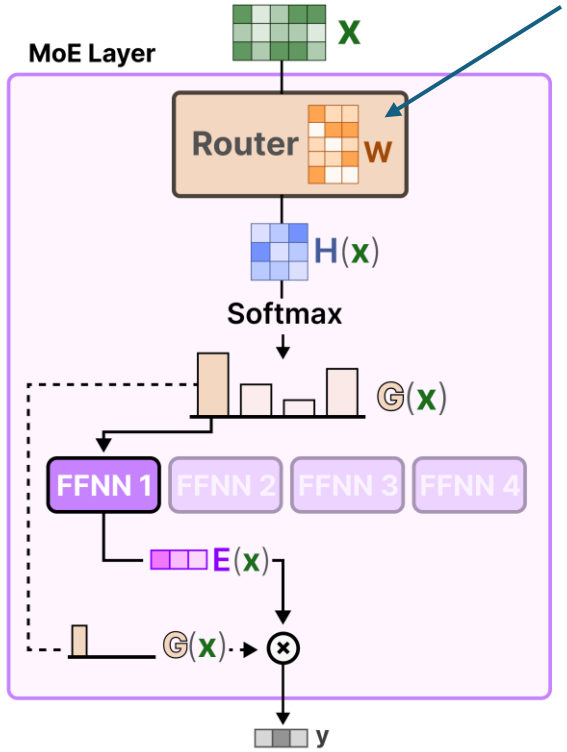

Loss vs Tokens
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
Loss
0
2
4
6
8
10
12
14
16
Tokens (Trillion)

source

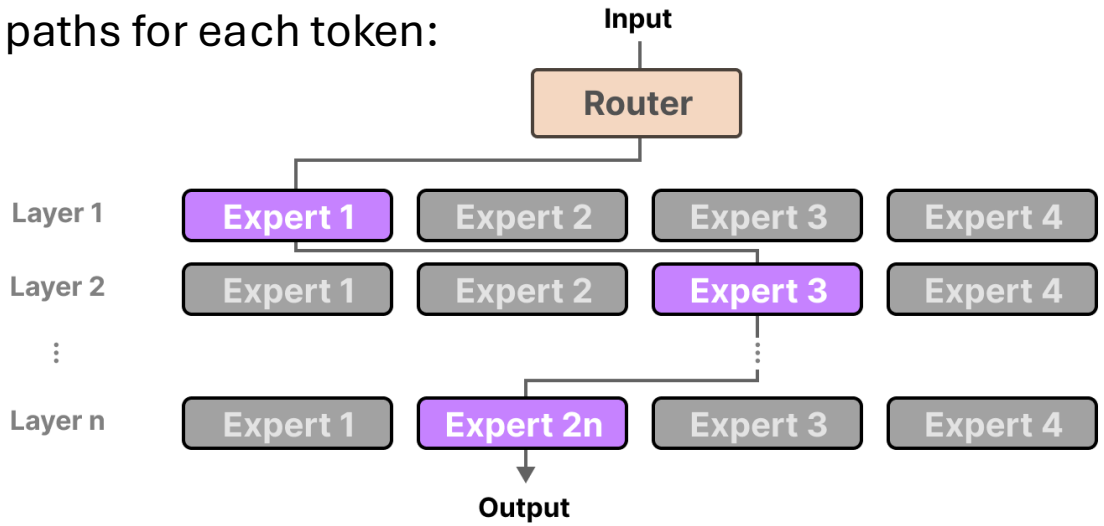
Mixture-of-Experts (MoE)

idea: replace big FFNN with several smaller ones and activate only few
→ less active parameters

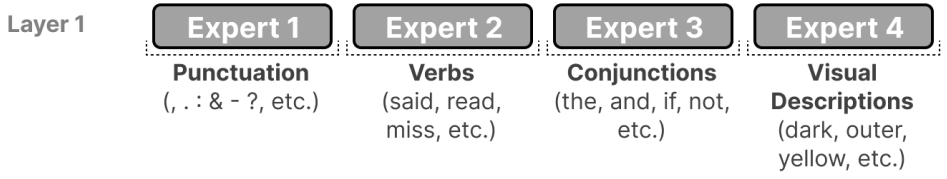


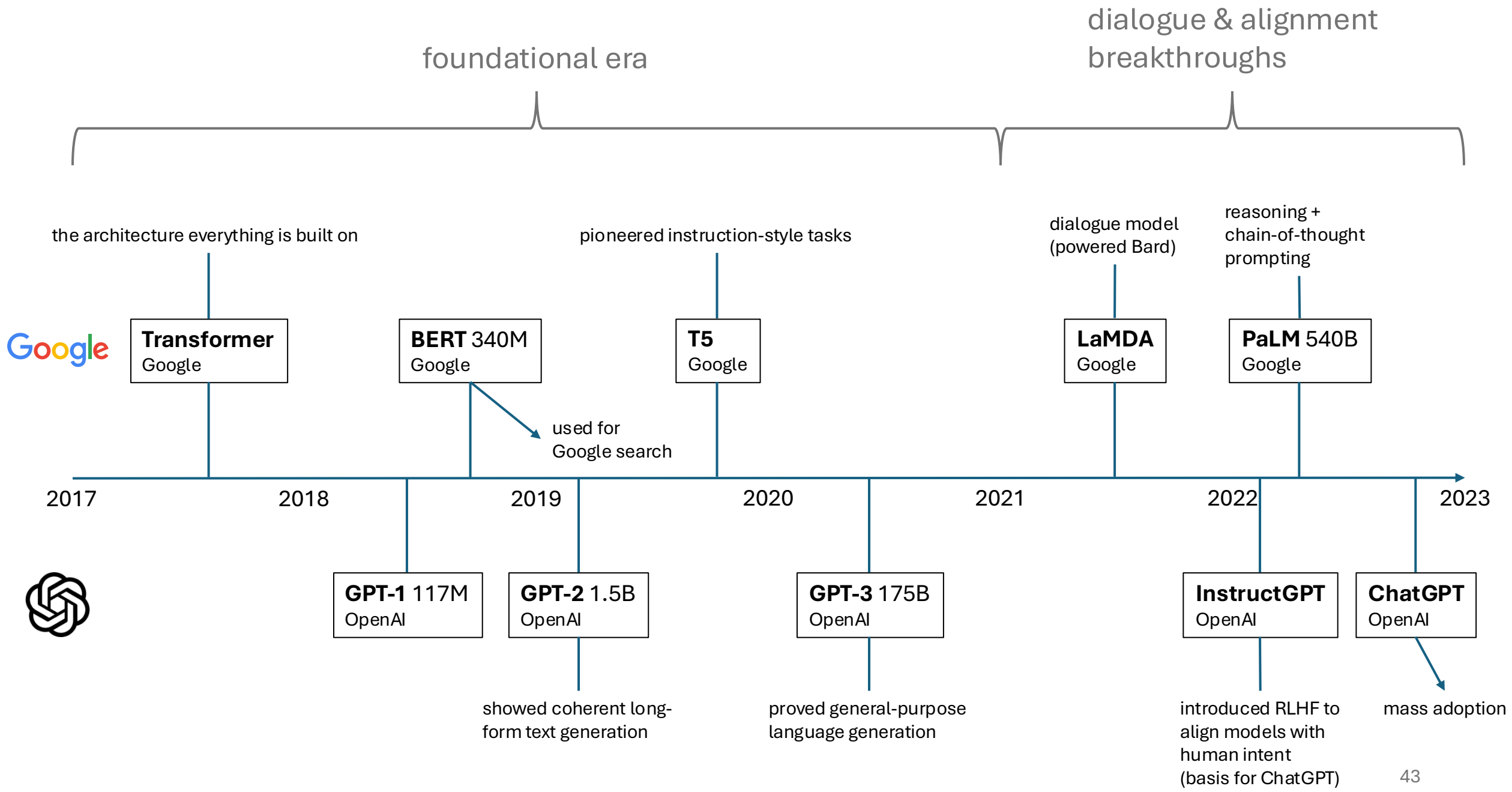


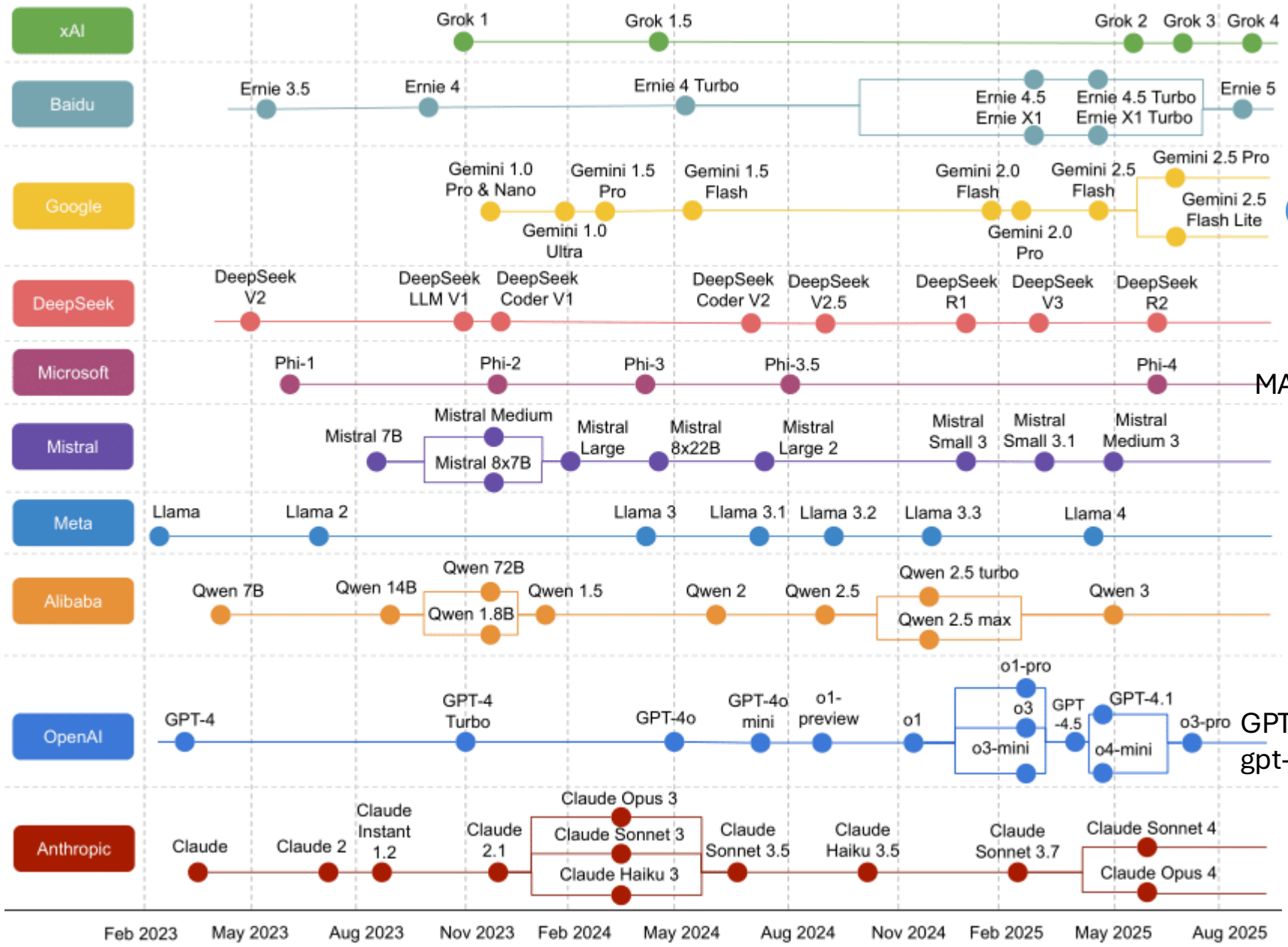
different paths for each token:



example (actually more abstract):




foundational era
dialogue & alignment breakthroughs
the architecture everything is built on
pioneered instruction-style tasks
dialogue model (powered Bard)
reasoning + chain-of-thought prompting
Transformer Google
BERT 340M Google
used for Google search
T5 Google
LaMDA Google
PaLM 540B Google
2017
2018
2019
2020
2021
2022
2023
GPT-1 117M OpenAI
GPT-2 1.5B OpenAI
GPT-3 175B OpenAI
InstructGPT OpenAI
ChatGPT OpenAI
showed coherent long-form text generation
proved general-purpose language generation
introduced RLHF to align models with human intent (basis for ChatGPT)
mass adoption



generative chatbots

Gemini

- evolved from LaMDA & PaLM
- integrated Google ecosystem

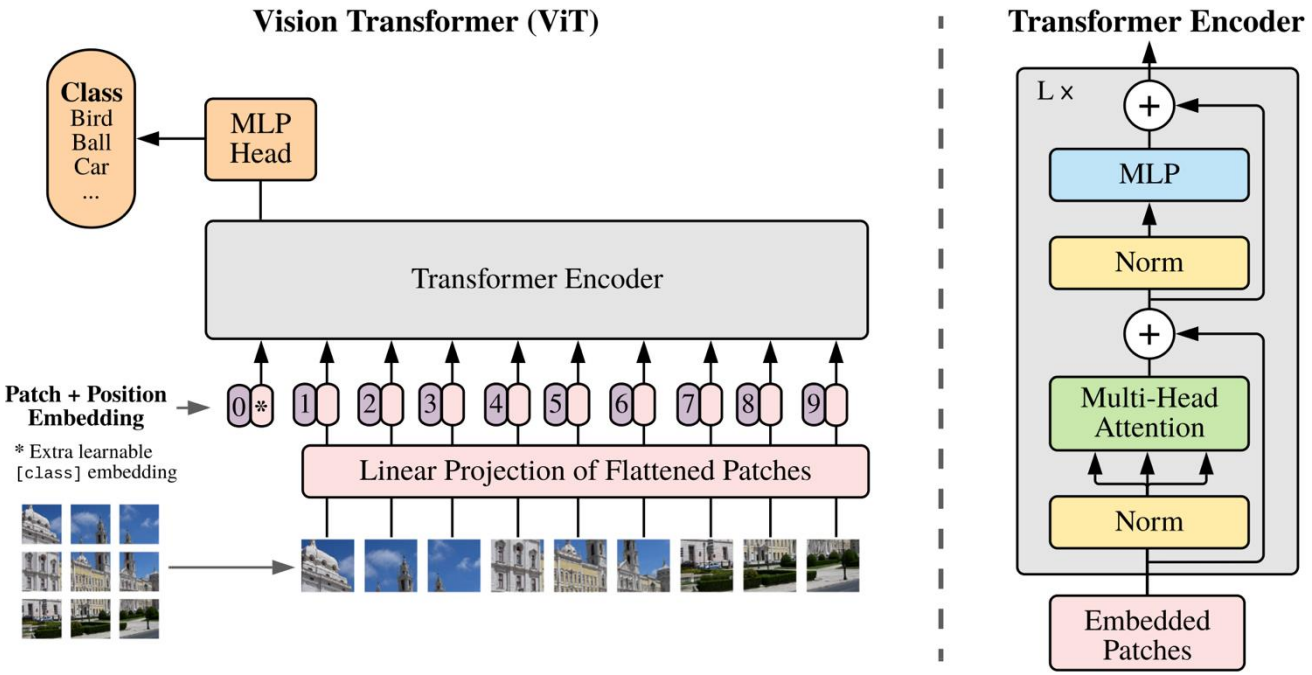
open-source models

ChatGPT

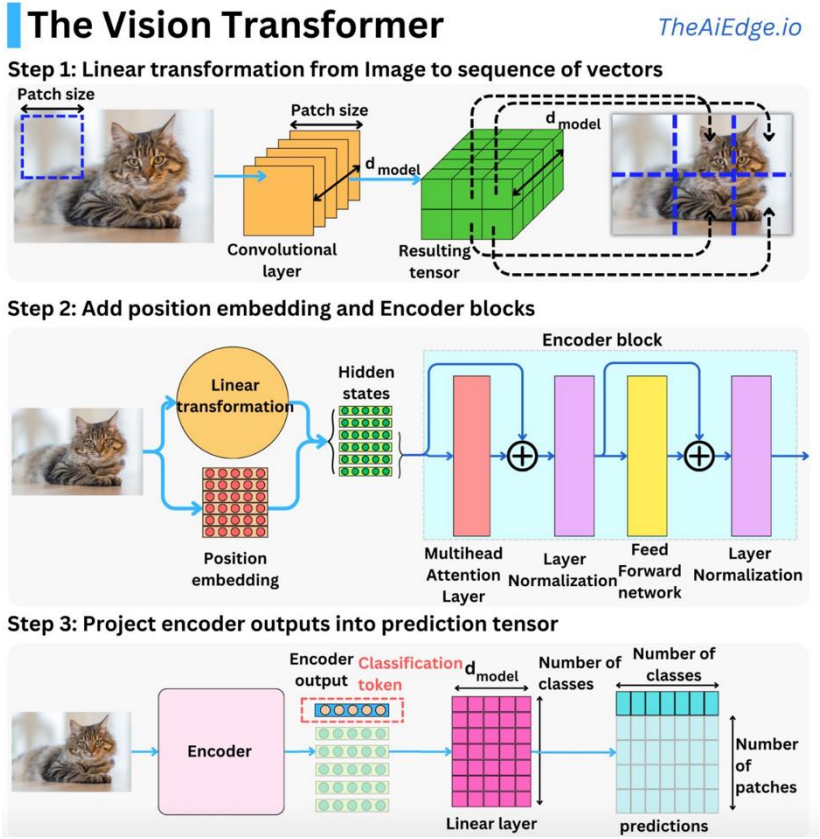
Claude

- different alignment strategy
- tailored for business users

Image Classification with Vision Transformer



source



formulation as sequential problem: split image into patches (tokens) and flatten, add positional embeddings

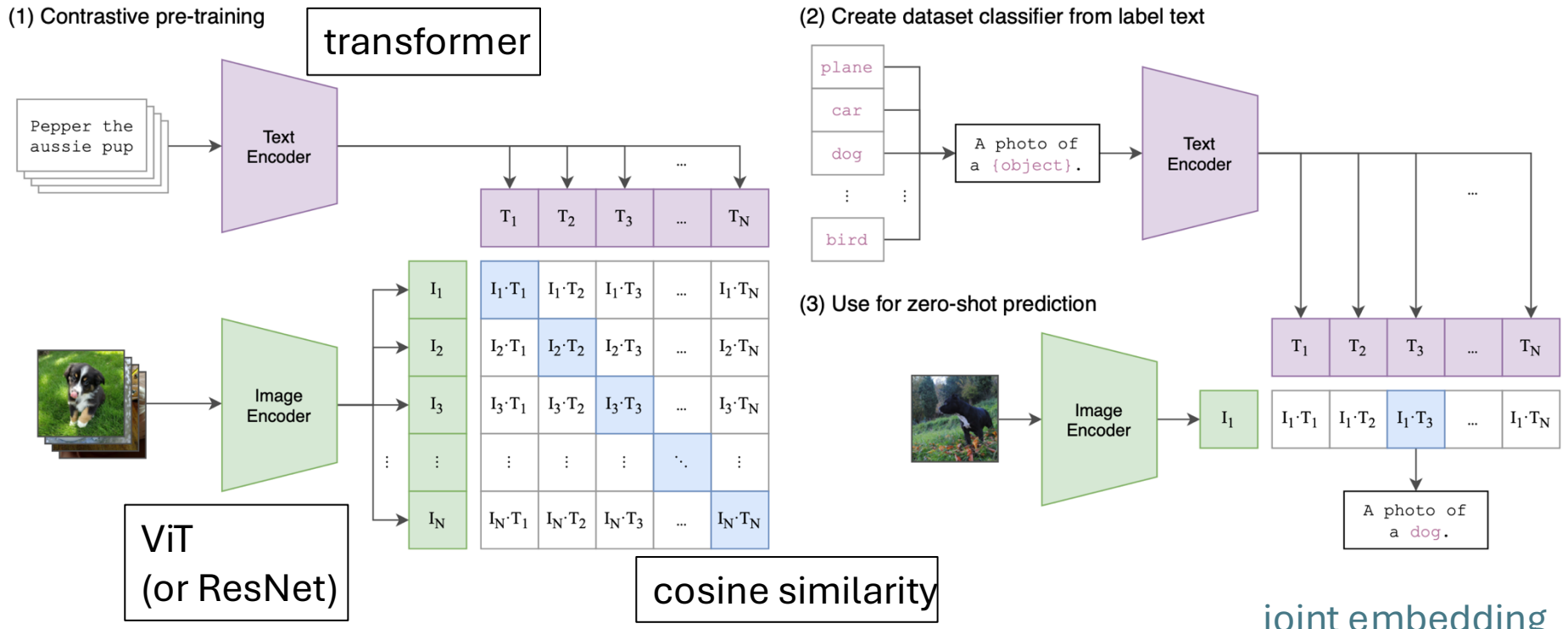
processing by transformer encoder: pretrain with image labels, fine-tune on specific data set

Combination of Vision and Text

example: CLIP (Contrastive Language-Image Pre-training)

learn image representations by predicting which caption goes with which image (pretraining)

→ zero-shot transfer (e.g., for image classification)



joint embedding

Modern Language Models in a Nutshell

tokenization: split text into chunks

pretraining: self-supervised, next- or masked-token prediction

- vector embeddings from tokens (semantic meaning)
- add positional encoding (attention permutation-invariant)
- attention mechanism (transformer) → contextual embeddings

finetuning: specialization

classification model (specific task):
finetuning on labeled data set (often using encoder-only transformer)

assistant/chatbot (multipurpose):
instruction tuning (decoder-only transformer), potentially followed by alignment

Transfer Learning from Foundation Models

idea:

- pretrain a big model (called foundation model) on a broad data set
- then use these learnings for subsequent trainings on specific (typically narrow) data by means of finetuning or feature extraction

→ here: self-supervised, internet-scale pretraining of models like BERT or GPT

works well for homogenous, unstructured data (e.g., text, images)
but very difficult for heterogenous, structured/tabular data

Finetuning of BERT

- get pretrained BERT as foundation model (MLM pretraining: loss computed against projection from last hidden state of [MASK] token to vector of logits)

- remove pretraining classification head (→ get contextual token embeddings)
- add another, task-specific classification head (see a few examples on the following slides)

- train (finetune) with data for actual task at hand (starting with random weights for new classification head and pretrained weights for the rest)
- options: finetune all pretrained weights or only a part (freezing the others)

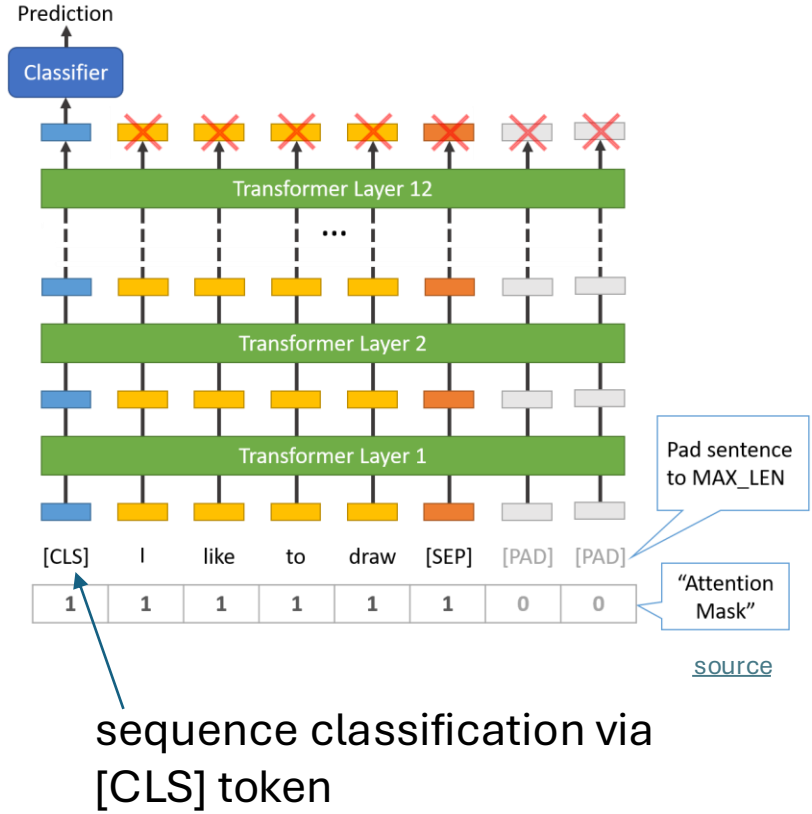
Example: Sequence Classification

e.g., classify sentiment or topic

- a special [CLS] token is prepended to the input (already present in pretraining)
- its final hidden state (vector embedding) is passed into a classification head (resulting in logits over different categories)

→ [CLS] embedding is finetuned to summarize the whole sequence

(alternative without [CLS]: average pooling)



Example: Sentence Embeddings

use case: sentence similarity / retrieval (see RAG)

BERT optimized for NSP, not for mapping independent sentences into a comparable vector space → need to finetune on considered similarity label

- paraphrase (yes or no)
- natural language inference (entailment/contradiction/neutral)
- semantic textual similarity (human-judged similarity score) → regression loss

approach with vanilla BERT:

- input: [CLS] Sentence A [SEP] Sentence B [SEP]
- [CLS] token embedding as aggregate representation of the pair
- classifier/regressor head trained directly on considered similarity label

SBERT

issue with vanilla BERT:

prediction of similarity score requires each pair of sentences to be processed together (no independent sentence embeddings) → $O(n^2)$ for n sentences

SBERT: instead of finetuning vanilla BERT,

- uses two (or three) copies of same BERT model (with shared weights)
- passes each sentence through one of the copies to produce its embedding from last hidden states (use [CLS] token or average pooling)
- puts common head on top of copies, optimizing $\text{similarity}(u, v) \approx \text{label}(u, v)$ of sentence embeddings u and v (→ different SBERT versions for different similarity tasks/labels)

→ cosine similarity (or dot product) can then be used to compare independent sentence embeddings

Example: Token Classification

examples of use cases:

- part-of-speech tagging: assigning each word in a sentence its grammatical role (or part of speech) such as noun, verb, etc.
- named entity recognition: identifying and classifying named entities (proper nouns) in text into predefined categories like Person, Organization, Location, Date, etc.

per-token predictions needed → no pooling or [CLS] usage

instead, just project each token's hidden state independently into logits and sum the per-token losses in each sequence

Finetuning Scenarios

features more generic in early layers (e.g., token structure, syntax, part-of-speech) and more specific to the pretraining data in later layers (e.g., word sense, entity linking)

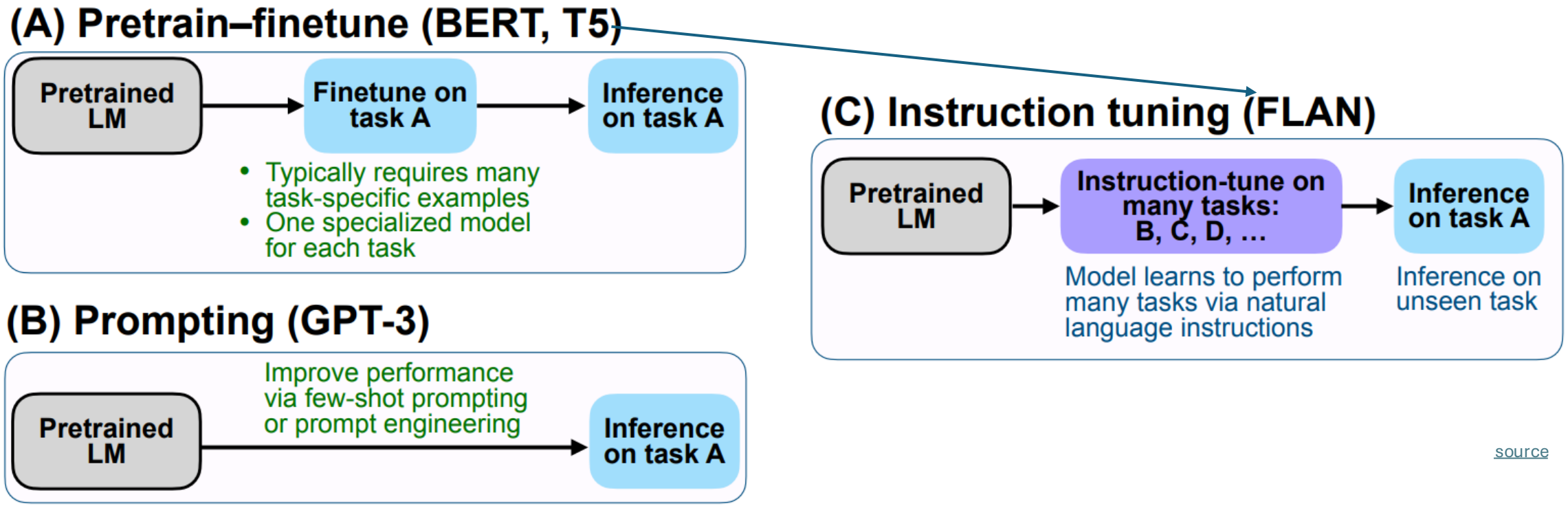
→for small finetuning data sets, typically keep weights of early layers fixed to avoid overfitting

→for large finetuning data sets (less danger of overfitting), finetuning all layers can be beneficial

Instruction Tuning

instruction tuning is a form of supervised finetuning (SFT)

so is the finetuning discussed before (although usually just called finetuning)



source

Instruction Tuning Data

instruction tuning = SFT with instruction–response data

Instruction: "Summarize this article in one sentence."

Output: "The article discusses the effects of climate change on polar bears."

instruction tuning data fed to model as input sequences, just like in pretraining

objective still next-token prediction (language modeling)

structure of text sequences only thing telling the model where instruction ends and response begins

also learns formatting: desired patterns embedded in instruction–response pairs

Base vs Instruction-tuned LLM

Explain why the sky is blue, as if you were talking to a curious 10-year-old.

base LLM response:

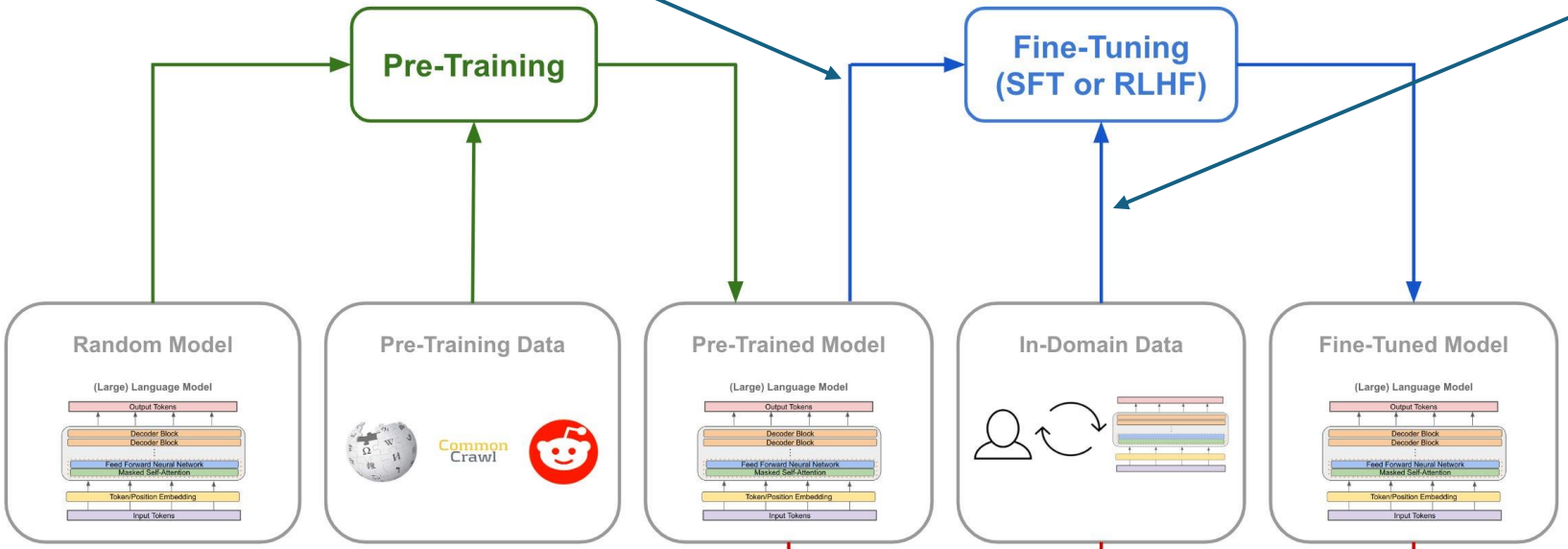
"The sky is blue because of the way light interacts with the atmosphere. Sunlight contains many colors, and when it passes through air molecules, the blue light scatters more than other colors. That's why we see the sky as blue during the day."

instruction-tuned LLM response:

"Imagine the sky is like a giant ocean of tiny invisible balls called air molecules. When sunlight shines through, the blue light bounces around more than the other colors, kind of like how tiny balls make waves in a pool. That's why, when you look up, the sky looks blue! And at sunset, the light has to travel through more air, so the reds and oranges shine through instead."

generic “instruct“ versions of LLMs
(potentially also aligned)
e.g., Llama-2-7b-chat, Mistral-7B-Instruct

expert LLMs



base LLM: poor instruction following,
no proper multi-turn converstations
(e.g., Llama-2-7b, Mistral-7B)

RAG

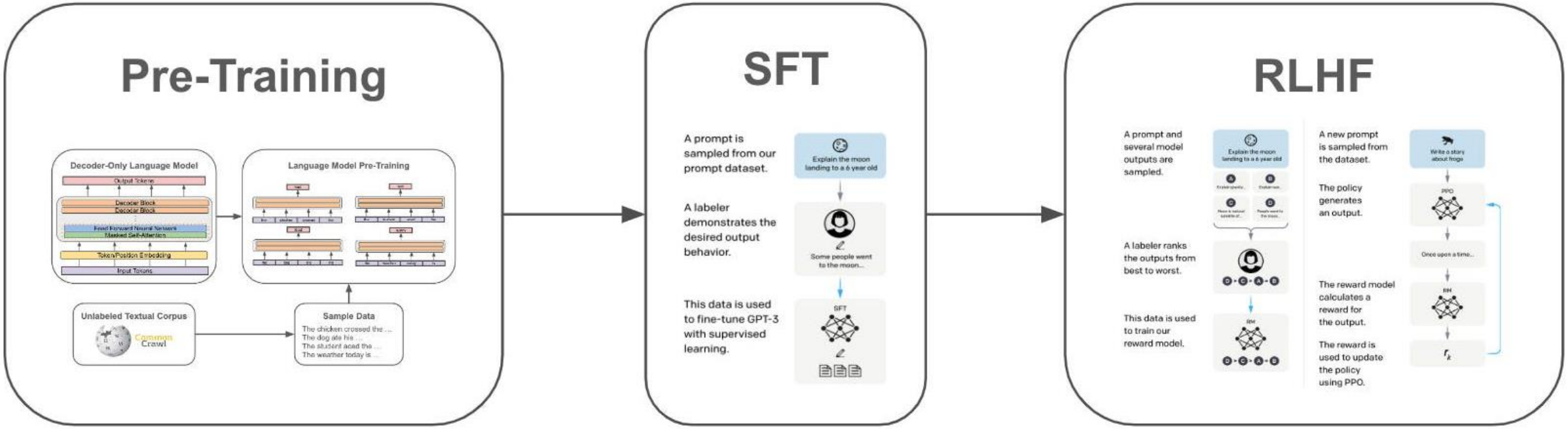
ready to become
an LLM chatbot

Use Cases of Instruction Tuning

- task specialization (e.g., code generation)
- domain-specific instructions (e.g., interpret molecular formulas)
- style or tone adaptation (e.g., corporate writing style, friendly chatbot tone)
- reasoning (CoT inputs) → reasoning models

Preference Alignment

RLHF can achieve both instruction following and preference alignment



here: GPT-3

source

idea: aligning LLM output with human preferences

InstructGPT: SFT followed by reinforcement learning from human feedback (RLHF)

GPT-3 → InstructGPT → ChatGPT

dialog system (aka conversational agent or chatbot): any system designed to communicate with humans using natural language

with instruction-tuned and aligned LLMs: assistant-style models

crucial: multi-turn dialogue

→ passing conversation history as part of input to LLM at each step

lightweight PyTorch re-implementation of ChatGPT: nanochat

Prompt Engineering

prompt: user interface to the model

prompt engineering: shape model behavior using well-structured prompts

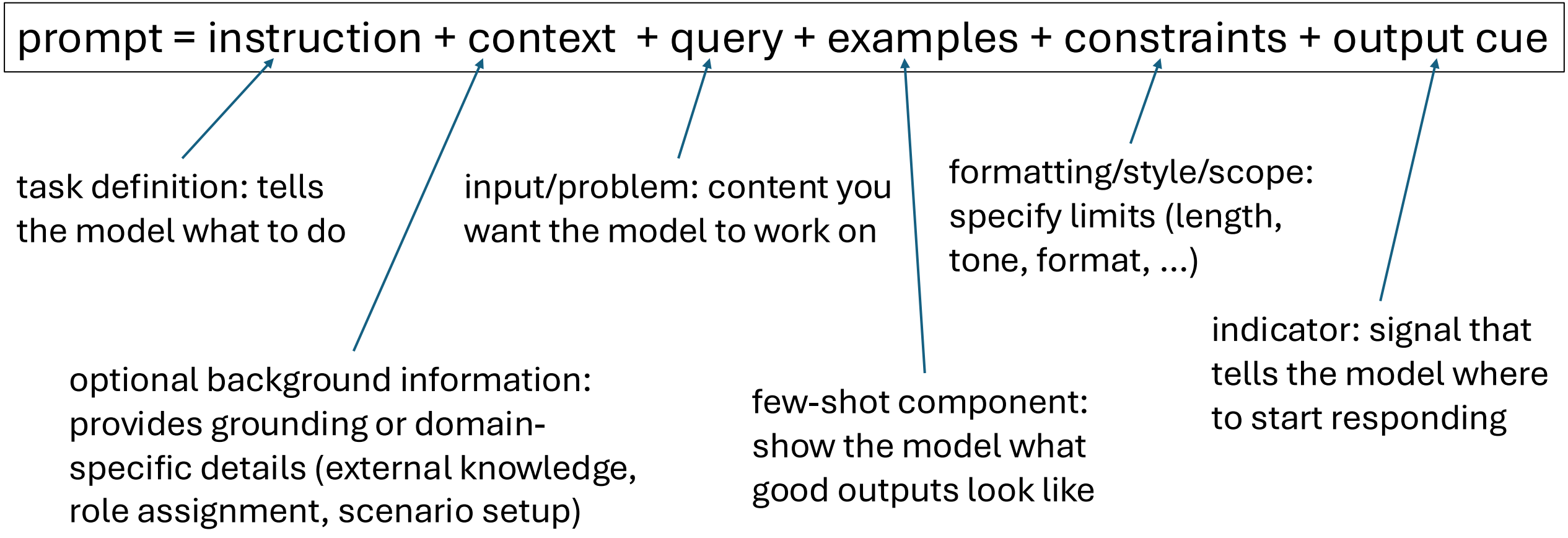
difference from traditional user interfaces:

probabilistic, not deterministic (shape the *distribution* of the model's responses)

→ good prompts can improve a model's performance and reduce hallucination, bias, or inefficiency

prompt engineering, GPT guide

Typical Prompt Structure



Example Prompt

Explain the following statement in three sentences.
Text: "Large language models show emergent abilities when scaled."
Answer:

- instruction: *Explain the following statement in three sentences.*
- query (input/problem): *Large language models show emergent abilities when scaled.*
- constraint: implicit → *in three sentences*
- output cue: *Answer:*

Simplest Possible Prompt

often query phrased as instruction:

```
Explain: Large language models show emergent abilities when scaled.
```

but without instruction, model just completes/continues the query text (usually not really useful):

```
Large language models show emergent abilities when scaled.
```

At the bare minimum, you only need the query. But for reliable, controllable outputs, you usually need to scaffold it with instruction, context, constraints, etc.

Best Practices: Clear Instruction & Query

Tell me about photosynthesis.

better:

Explain photosynthesis in simple terms suitable for a 12-year-old, using a short paragraph and one analogy.

→ clearly state what you want the model to do (best when specific, unambiguous, and outcome-focused)

Best Practices: Give Context

Write about the causes of the French Revolution.

better:

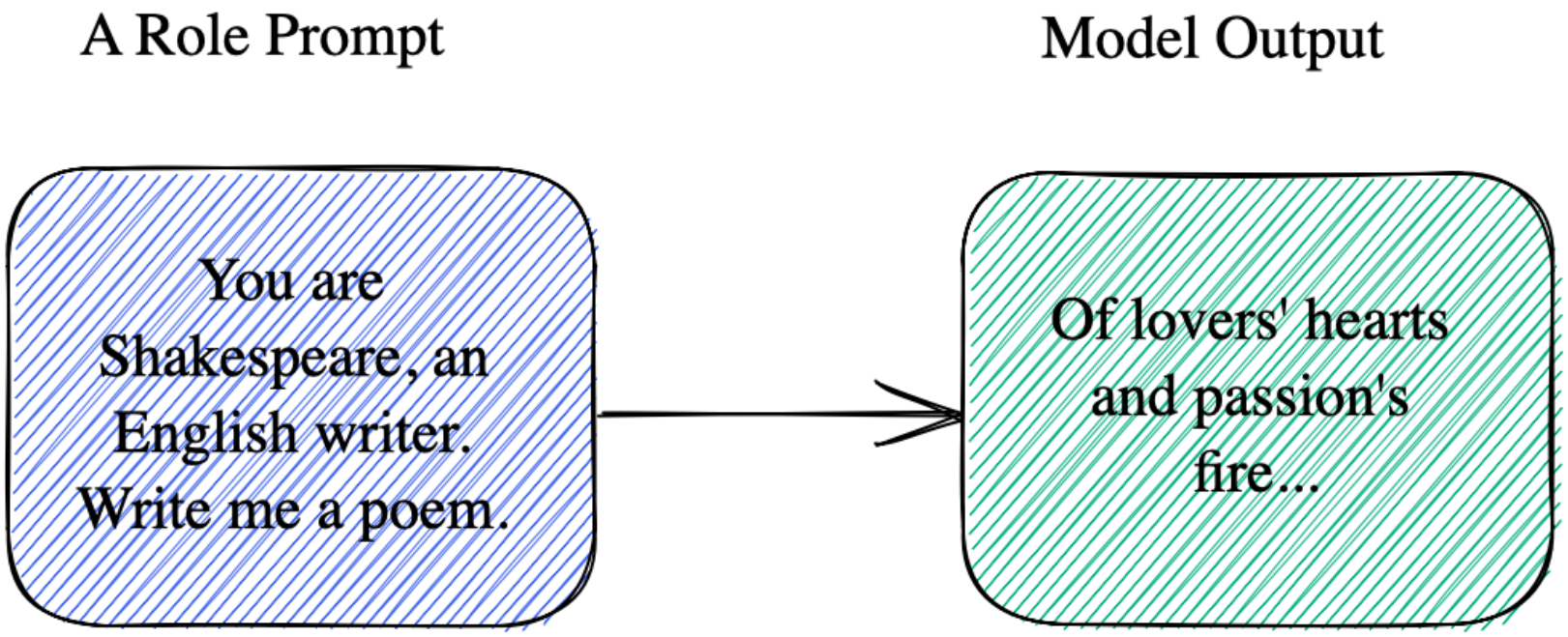
You are a historian writing a study guide for undergraduate students.

Write a concise explanation of the causes of the French Revolution, highlighting 3 major factors.

context: *You are a historian writing a study guide for undergraduate students*.

→ role assignment and audience specification steer tone and depth

Aka Role Prompting



Best Practices: Specify Constraints

List some facts about Mars.

better (if you want specific outputs):

List exactly 3 scientifically verified facts about Mars.

Output them in JSON format with keys: "fact1", "fact2", "fact3".

Output Indicators

earlier models (pre-ChatGPT) were very sensitive to formatting (training datasets contained clear separators)

→ certain cues like *Answer:*, *A:*, *Response:*, or even ### dramatically affected performance (better alignment with training data)

not strictly required anymore for modern instruction-tuned LLMs

but can still be useful for automation or complex prompts (improve consistency, reduce ambiguity):

- classification: Label:
- code generation: Code:

Examples/Demonstrations

Convert these temperatures from Celsius to Fahrenheit:

25, 30, 35

few-shot examples help for unusual, complex, or under-specified tasks:

Convert these temperatures from Celsius to Fahrenheit:

C: 0 → F: 32

C: 10 → F: 50

C: 20 → F: 68

Now continue for:

C: 25, 30, 35

Important Subtlety: Formatting & Abstraction

minimum effect of examples: enforce formatting patterns (small models)

C: 25 → F: 666

maximum: approximate transformation rule behind examples (large models)

C: 25 → F: 77

But not even the largest LLMs actually derive $F = C \cdot 1.8 + 32$.

instead: interpolation from similar training cases memorized in model weights (not directly interpolating the numerical values, but its vector representations)

→ looks like reasoning, but is just probabilistic pattern completion

→ no algorithmic guarantee (typically breaks for unusual inputs)

Structured Design

So, prompts aren’t random — they have designable components.

different tasks emphasize different sections:

- creative writing → role/context
- (approximate) reasoning → examples
- structured output → constraints

Interfaces to Chatbots

UI

- web/mobile apps
- messaging platforms (Teams, WhatsApp, …)
- voice interfaces (Alexa, Siri, …)

API (typically REST / HTTP endpoints)

- example: OpenAI's `/completions`
- send structured request (usually JSON) with prompt and receive structured response
- enables integration into apps, websites, backend services

Example request gpt-3.5-turbo-instruct curl

```
curl https://api.openai.com/v1/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-3.5-turbo-instruct",
    "prompt": "Say this is a test",
    "max_tokens": 7,
    "temperature": 0
  }'
```

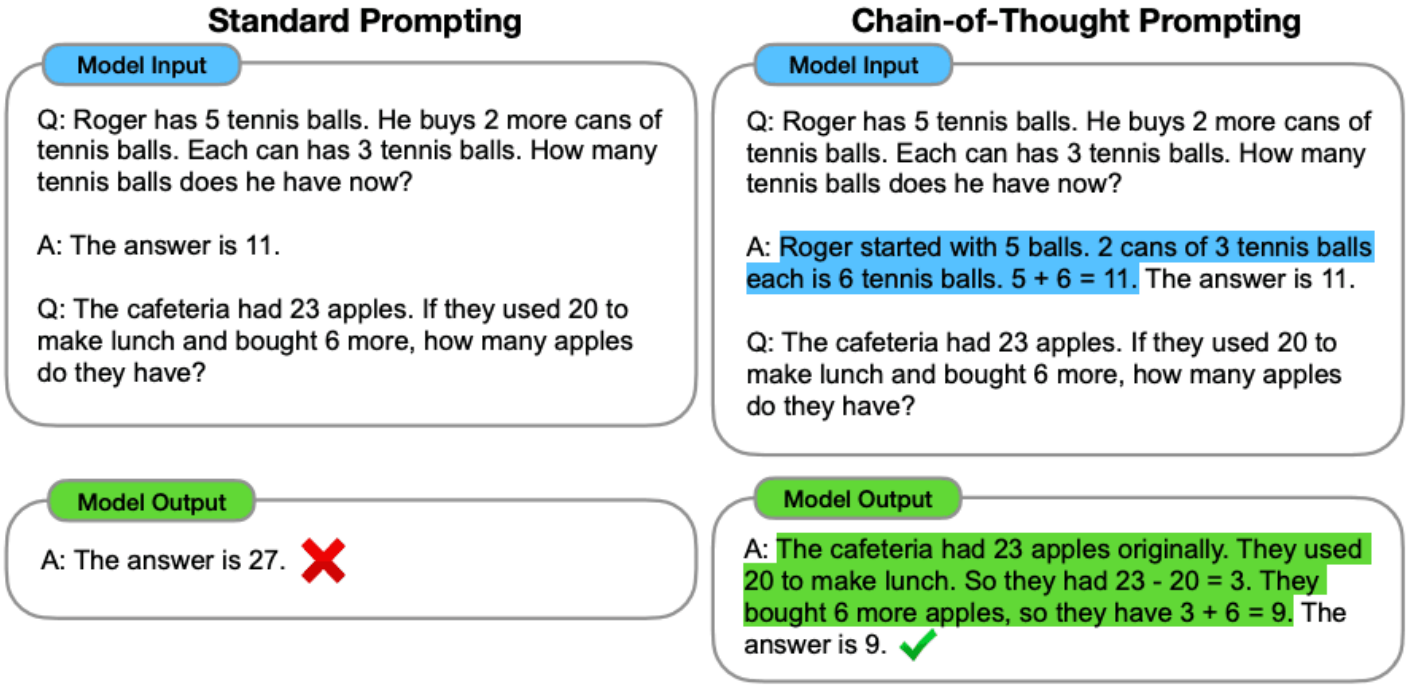
Response

```
{
  "id": "cmpl-uqkvlQyYK7bGYrRHQ0eXlWi7",
  "object": "text_completion",
  "created": 1589478378,
  "model": "gpt-3.5-turbo-instruct",
  "system_fingerprint": "fp_44709d6fcb",
  "choices": [
    {
      "text": "\n\nThis is indeed a test",
      "index": 0,
      "logprobs": null,
      "finish_reason": "length"
    }
  ],
  "usage": {
    "prompt_tokens": 5,
    "completion_tokens": 7,
    "total_tokens": 12
  }
}
```

Chain-of-Thought Prompting (CoT)

explicitly ask the model to **reason step by step** before giving a final answer

originally, by providing at least one reasoning example:

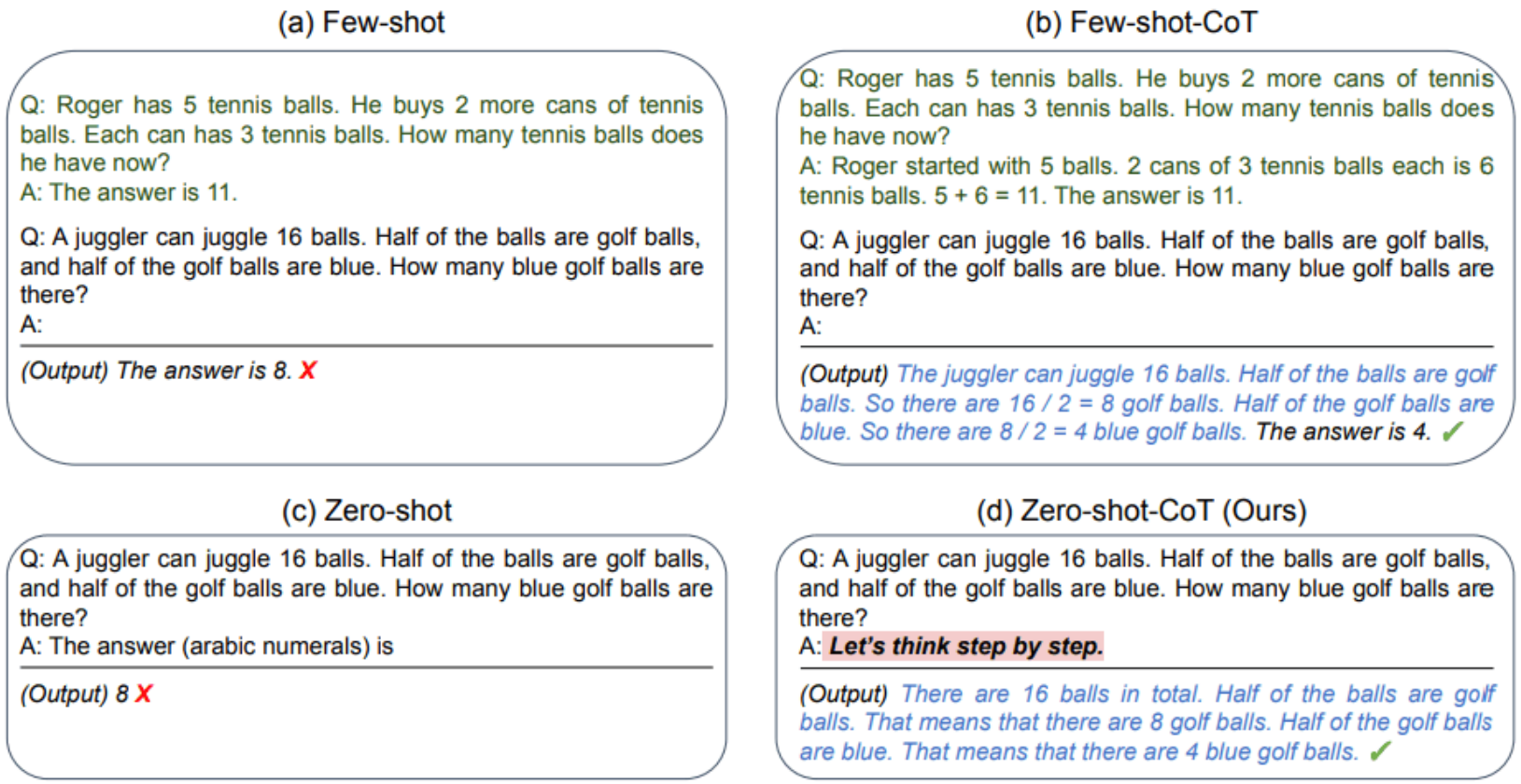


makes model outputs more legible

but no genuine interpretability of underlying LLM reasoning processes

Zero-Shot CoT

also works without examples, just through clever instructions ...



source

Hallucinations

knowledge encoded implicitly in weights rather than as explicit facts

→ probabilistic pattern completion can produce hallucinated facts

Instruction: What are Thomas Edison's main contributions to science and technology?

Response: Thomas Edison developed the first practical telephone and invented the light bulb.

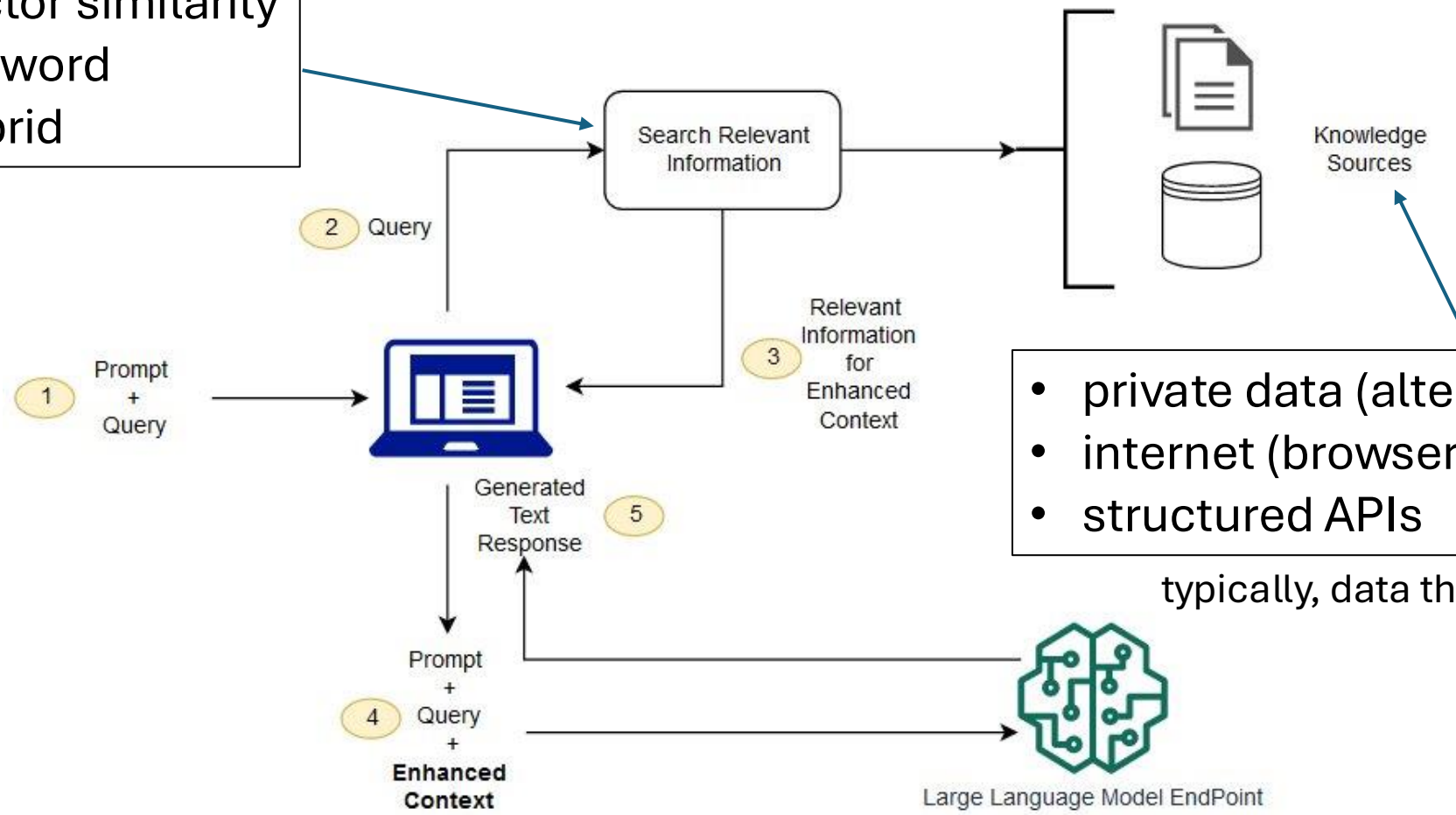
The response is factually incorrect. In reality, **Edison improved the light bulb, building on earlier designs by others**, and **Alexander Graham Bell invented the telephone**.

can be mitigated by retrieving the corresponding information from an external knowledge base and pasting it in context

(prompt engineering → context engineering)

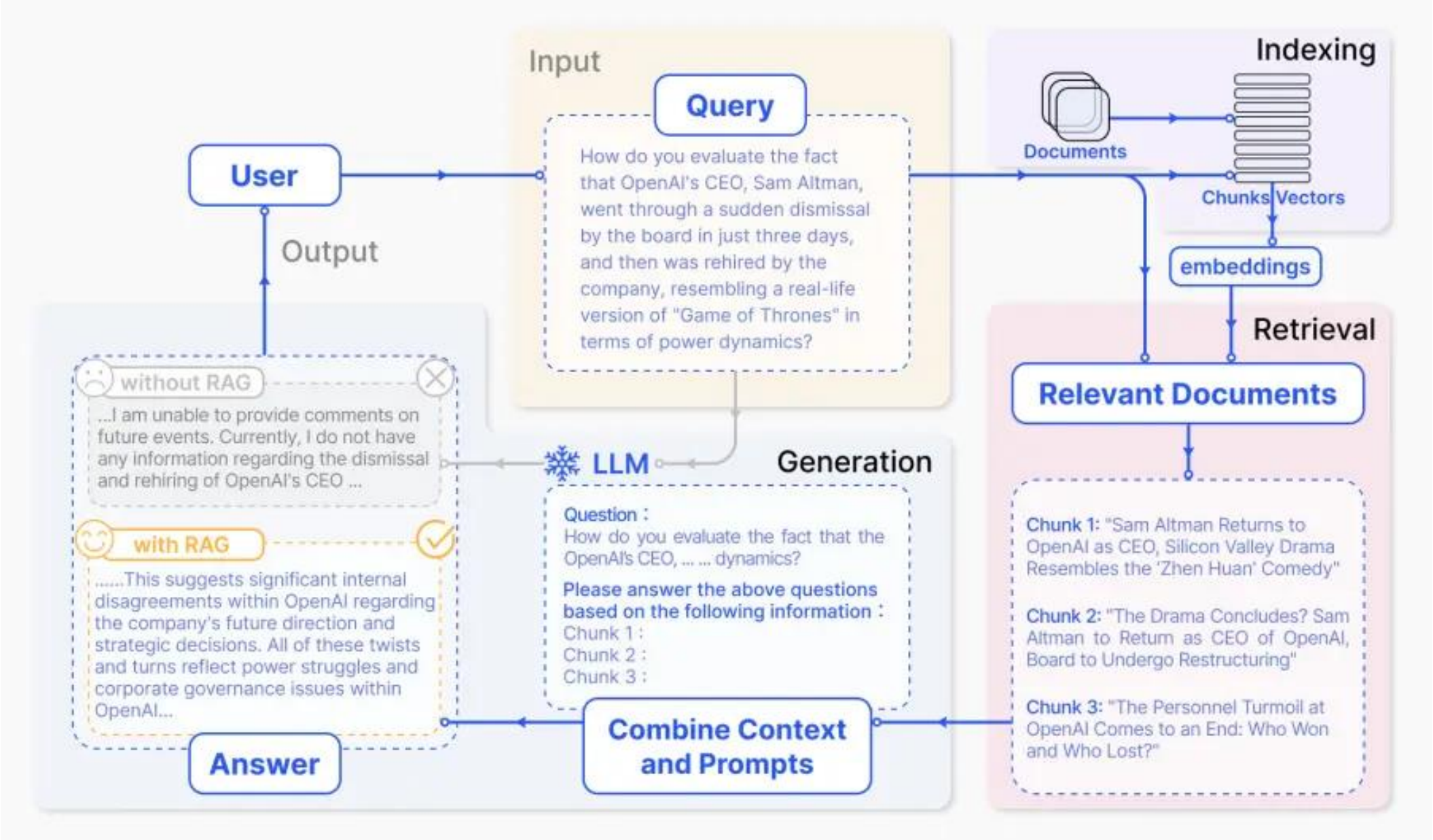
Retrieval Augmented Generation (RAG)

- vector similarity
- keyword
- hybrid



- private data (alternative to finetuning)
- internet (browser as tool)
- structured APIs

typically, data the model wasn‘t trained on


Input
Query
How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?
User
Output
Indexing
Documents
Chunks
Vectors
embeddings
Retrieval
Relevant Documents
Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"
Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"
Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"
LLM
Generation
Question：
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information：
Chunk 1：
Chunk 2：
Chunk 3：
Combine Context and Prompts
without RAG
...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...
with RAG
.....This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...
Answer

source

Embeddings & Vector Similarity Search

RAG approach:

- embed your documents (or passages) into a vector space (typically vector databases like FAISS, Pinecone, Weaviate, etc.)
- embed your query in the same space
- use a vector similarity search (like cosine similarity or dot product) to retrieve the most relevant documents
- feed those retrieved documents into your LLM to generate an answer

you generally don't embed word-by-word → sentence embeddings

- idea: average pooling over contextual token embeddings of transformer
- better: models specifically finetuned for semantic similarity (e.g., SBERT)

Example: sentence-transformers

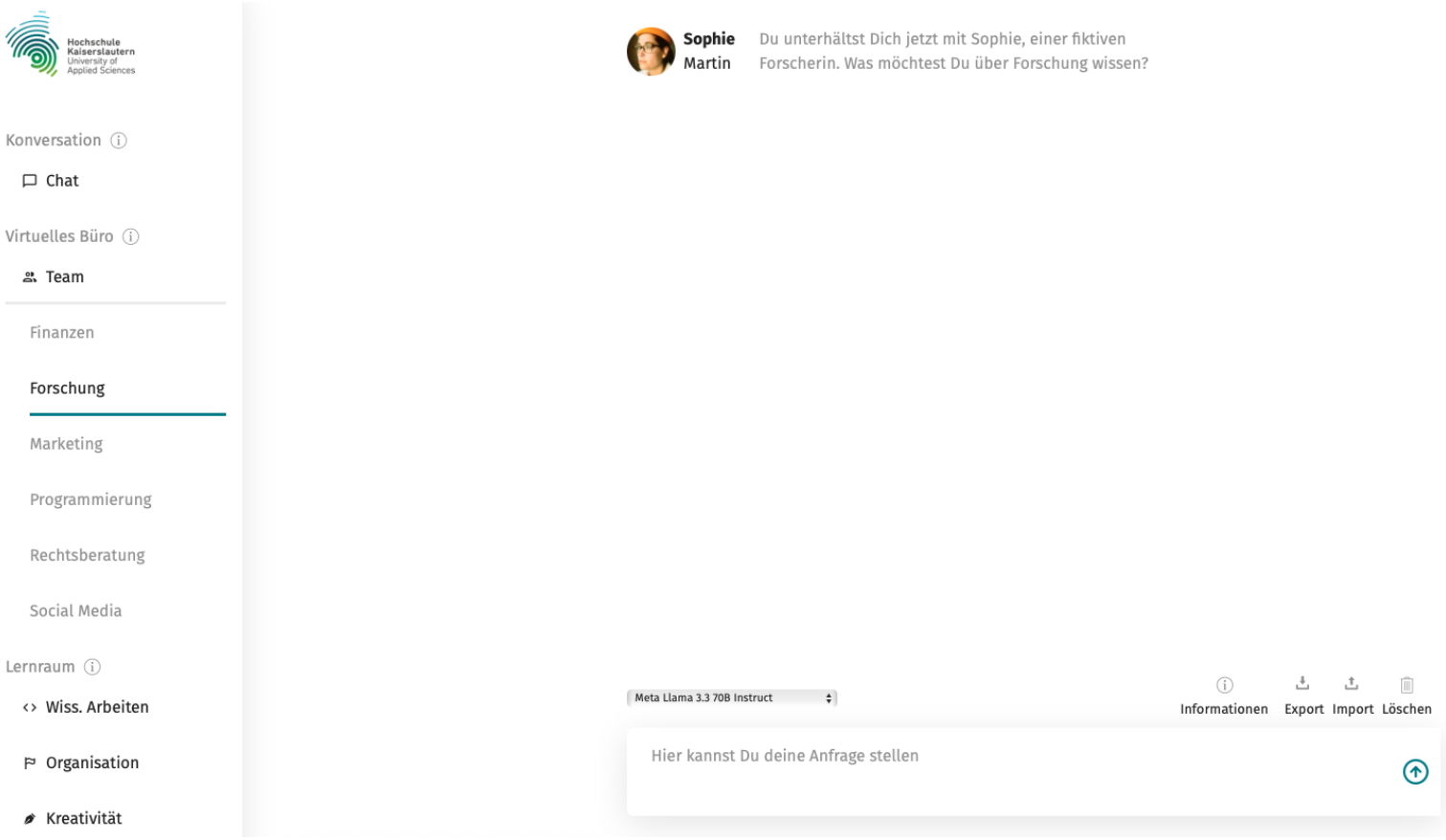
```python
from sentence_transformers import SentenceTransformer

# load a model
model = SentenceTransformer("all-MiniLM-L6-v2")

# encode sentences
sentences = ["RAG uses embeddings.", "We retrieve relevant passages."]
embeddings = model.encode(sentences, normalize_embeddings=True)

print(embeddings.shape)  # (2, 384)
```

Advanced Prompting Techniques

- self-consistency: aggregation of multiple sampled CoT reasoning paths
- meta-prompts / prompt templates: using one prompt to generate another
- prompt chaining: e.g., 1: extract facts → 2: summarize → 3: critique
- scaffolding: multi-turn prompt orchestration, planning, subtask decomposition
- reflexion: model revisits its output, identifies errors or gaps, and revises
- automatic prompt generation and optimization (e.g., OPRO)
- role-playing and system prompts: e.g., teacher, critic, planner

System Prompts

EDU-KI Chat:

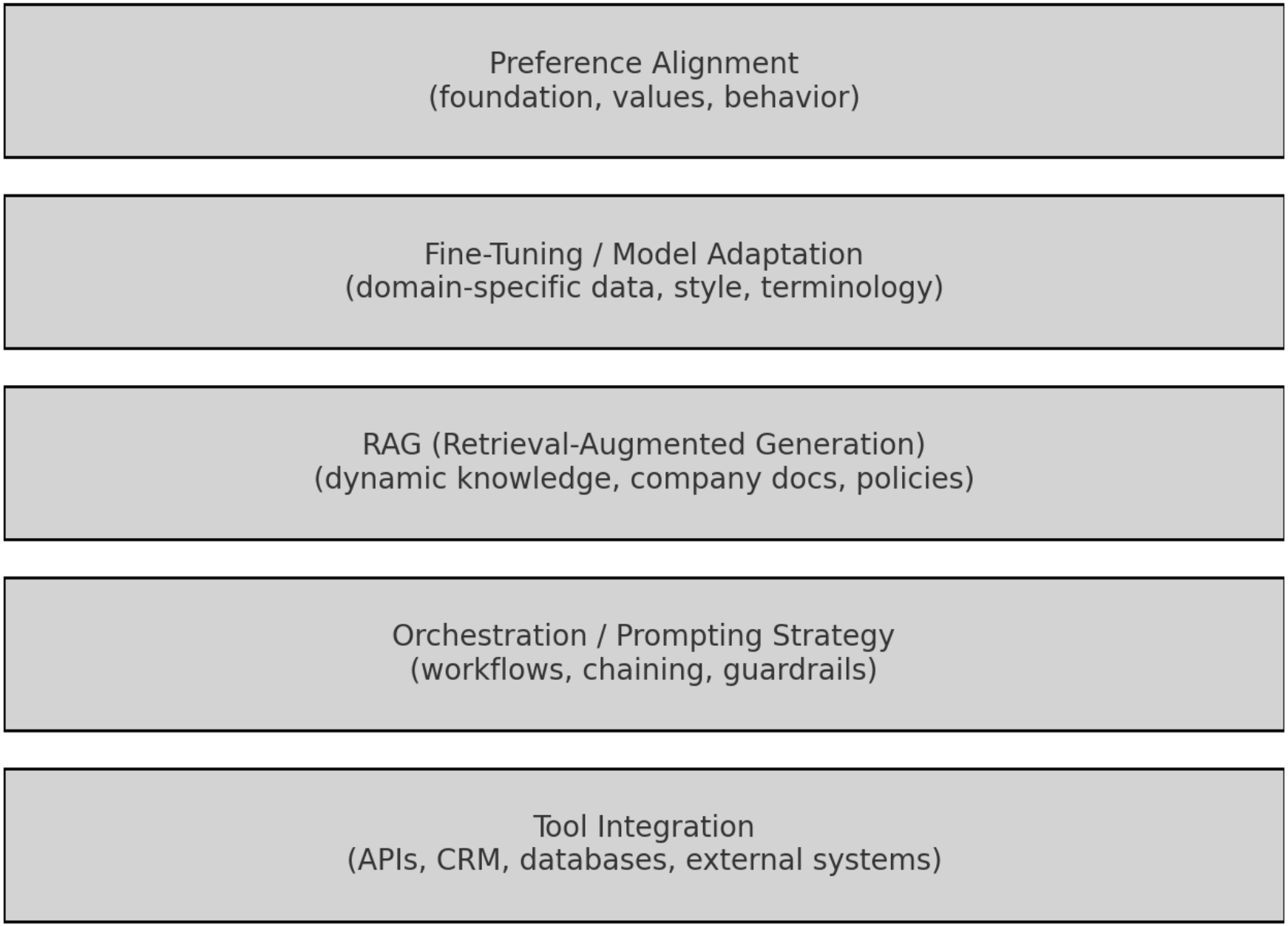


interaction framed by a role assignment prompt

typically set in the system prompt layer (not visible to the user)

→ user prompt is then interpreted within that predefined persona

Layers of Adapting a Chatbot to Company Needs



Rule of Thumb

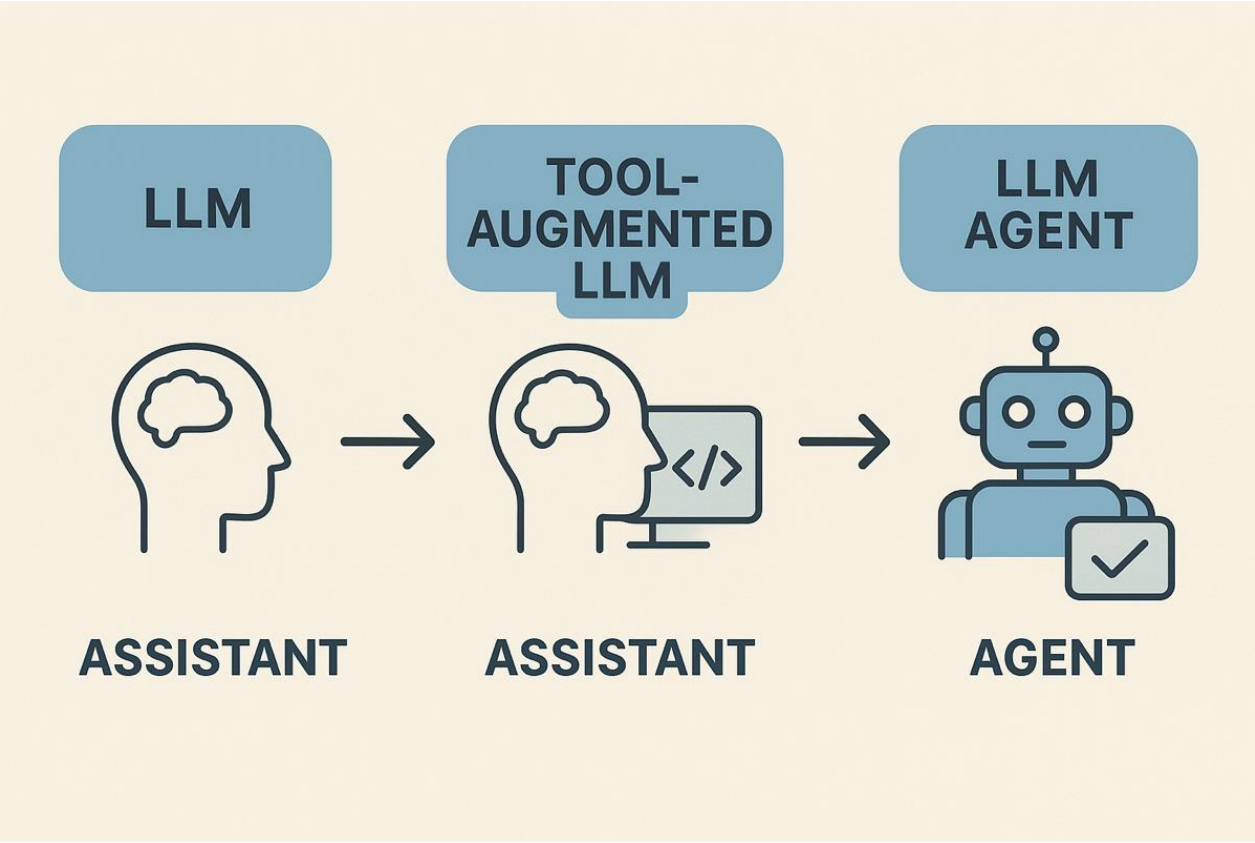
- **use prompting** → for small adjustments
- **use RAG** → for up-to-date knowledge, large knowledge bases, or private data
- **use finetuning** → for domain-specific expertise, custom style, or tasks that need the model's behavior to change fundamentally

Agents

latest AI evolutionary step: **LLMs as assistants** to **LLMs as agents**

passive, reactive

responding to user prompts

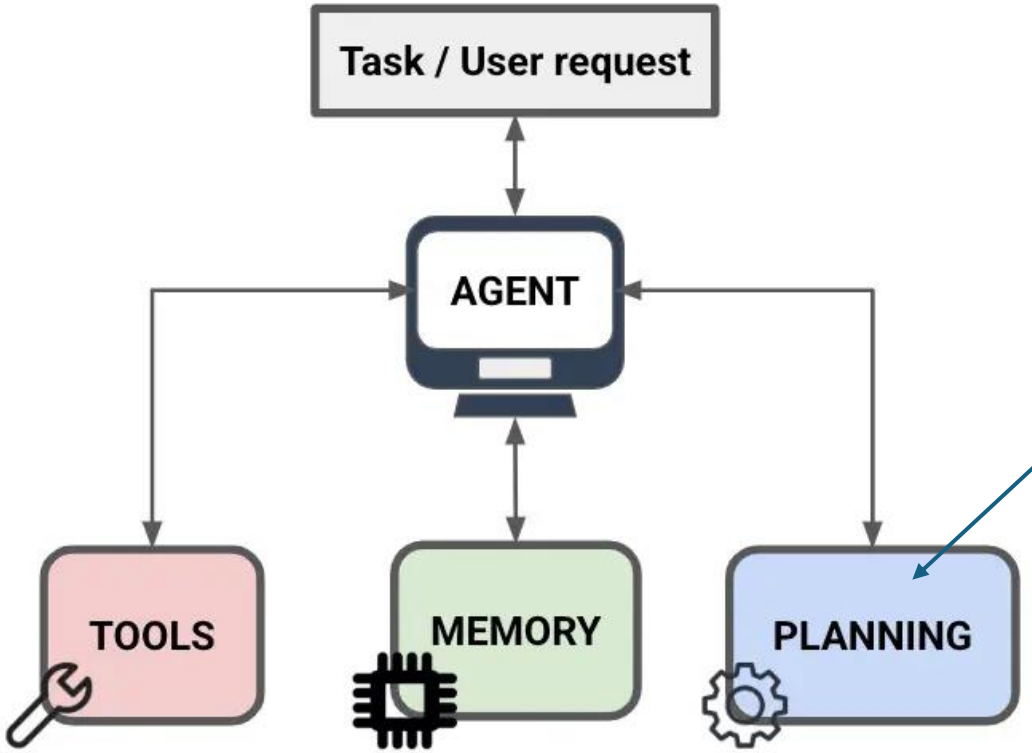


active, autonomous, decision-making entities

acting on goals or tasks

Agentic Intelligence

goal: autonomous problem-solving



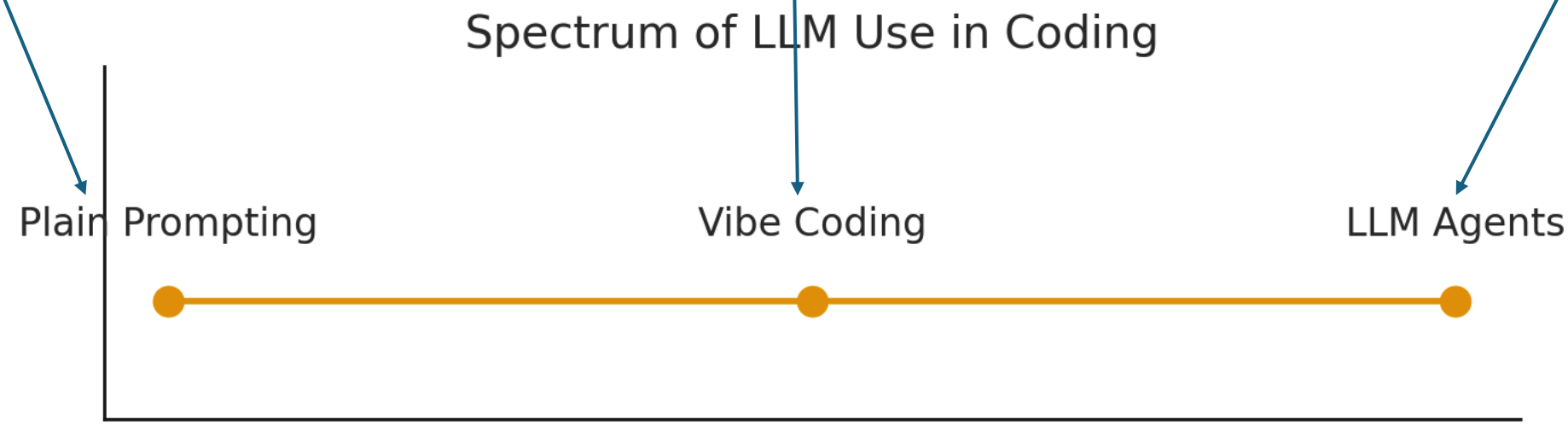
decomposition of complex task in simpler steps (uses reasoning capabilities)

Example: LLM Use in Coding

using ChatGPT or similar for snippets and Q&A

interactive pair-programming with tools like GitHub Copilot or Cursor

autonomous systems like AutoGPT plan and execute tasks with minimal human oversight



Intuitive and Analytical Thinking

two systems of human thinking (from Kahneman, *Thinking, Fast and Slow*):

- system 1: fast, automatic, intuitive, effortless, often subconscious
- system 2: slow, deliberate, analytical, effortful, conscious

LLMs analogous to system 1 (although without true intuition or understanding), lacking the depth of reflective, analytic reasoning

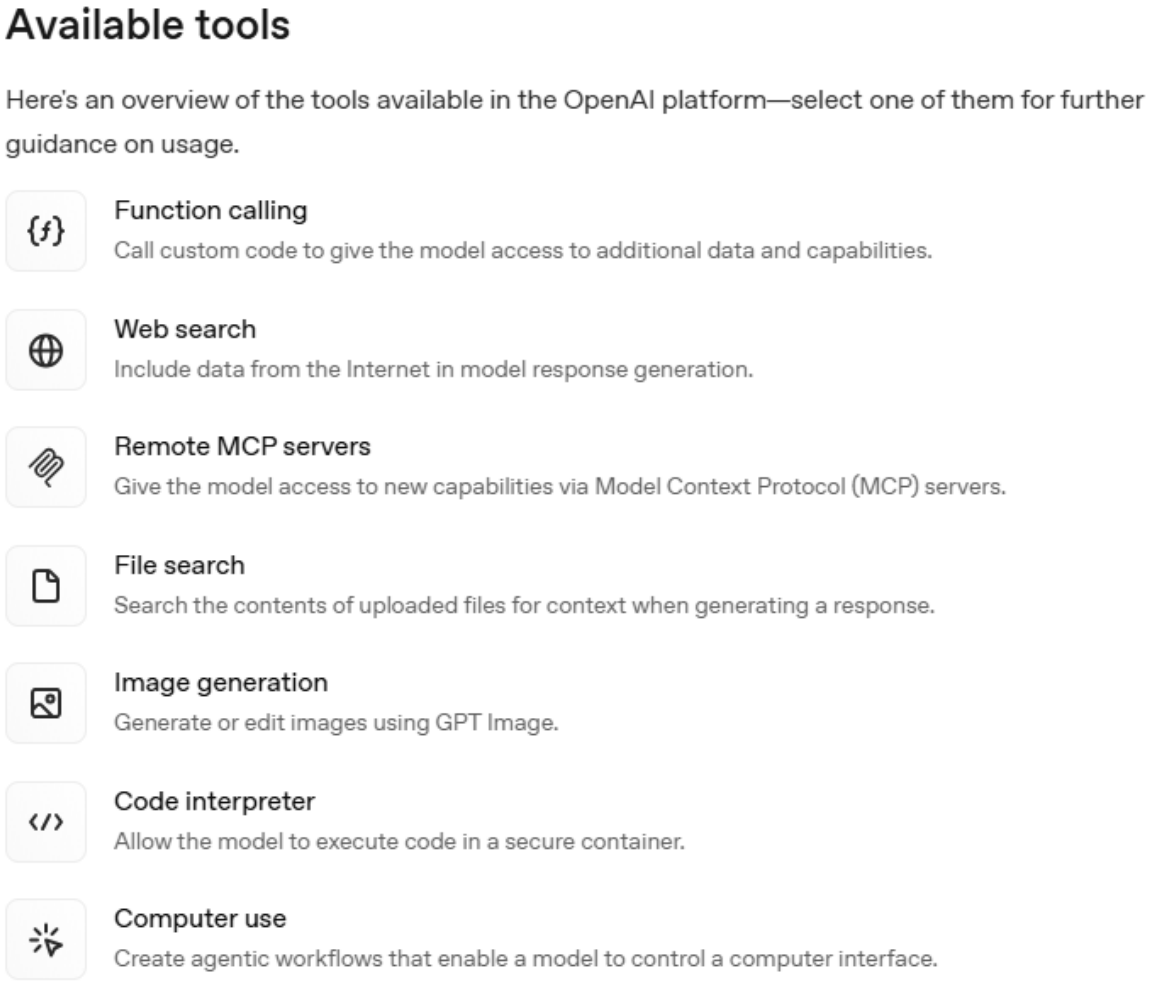
→ for agentic intelligence, need to enhance LLMs with system 2 capabilities, including tool usage and fact-checking

Tool Usage

crucial LLM capabilities:

1. know how to access different tools
   - code interpreters
   - web search
   - calculators (to overcome lack of mathematical understanding)
   - …
2. select right tool at right time

example: OpenAI



Acquistion of Tool Usage Skills

using instruction-tuned LLMs (like ChatGPT) as agent:

- core capability (*how* to use a tool) comes from pretraining and different finetuning stages
- tool access controlled at runtime via prompting and system messages
- tool selection (*when* to use *which* tool) decided from context

potentially, custom finetuning for specific tools or workflows

Test-Time Scaling

improve a model's performance (especially reasoning capabilities) without retraining it, by allocating more compute during inference time:

- more complex prompting strategies (CoT, self-consistency, prompt optimization, …)
- extensive tool use

→ test-time compute (number of samples, reasoning steps) as another scaling law (in addition to parameters, training data, training compute)

Reasoning Models

some reasoning capabilities:

standard LLMs using prompting strategies such as CoT or ReAct

→ reasoning emerges implicitly from pretraining (performance can be brittle and highly prompt-dependent)

better reasoning capabilities:

reasoning-specific LLMs explicitly trained or finetuned on reasoning objectives (often with CoT-style supervision or multi-step reasoning datasets)

→ learn to reason by default, not just when prompted

examples: OpenAI o1, DeepSeek-R1 (open source)

LLM Orchestration and Agents

wrappers around LLM calls to make them more useful in real-world applications

LLM orchestration:
pipelines + prompt templates + access to tools & memory + context management
→ structured environment
e.g., LangChain, LlamaIndex

agent systems:
LLM orchestration + autonomy/planning loop
→ kind of autopilot on top of orchestration layer
e.g., AutoGPT, OpenAgents, BabyAGI

Multi-Agent Systems

idea:

complex, distributed, and dynamic problems can be solved more efficiently, flexibly, and robustly by using multiple specialized agents rather than a single one

→ extended agent system:

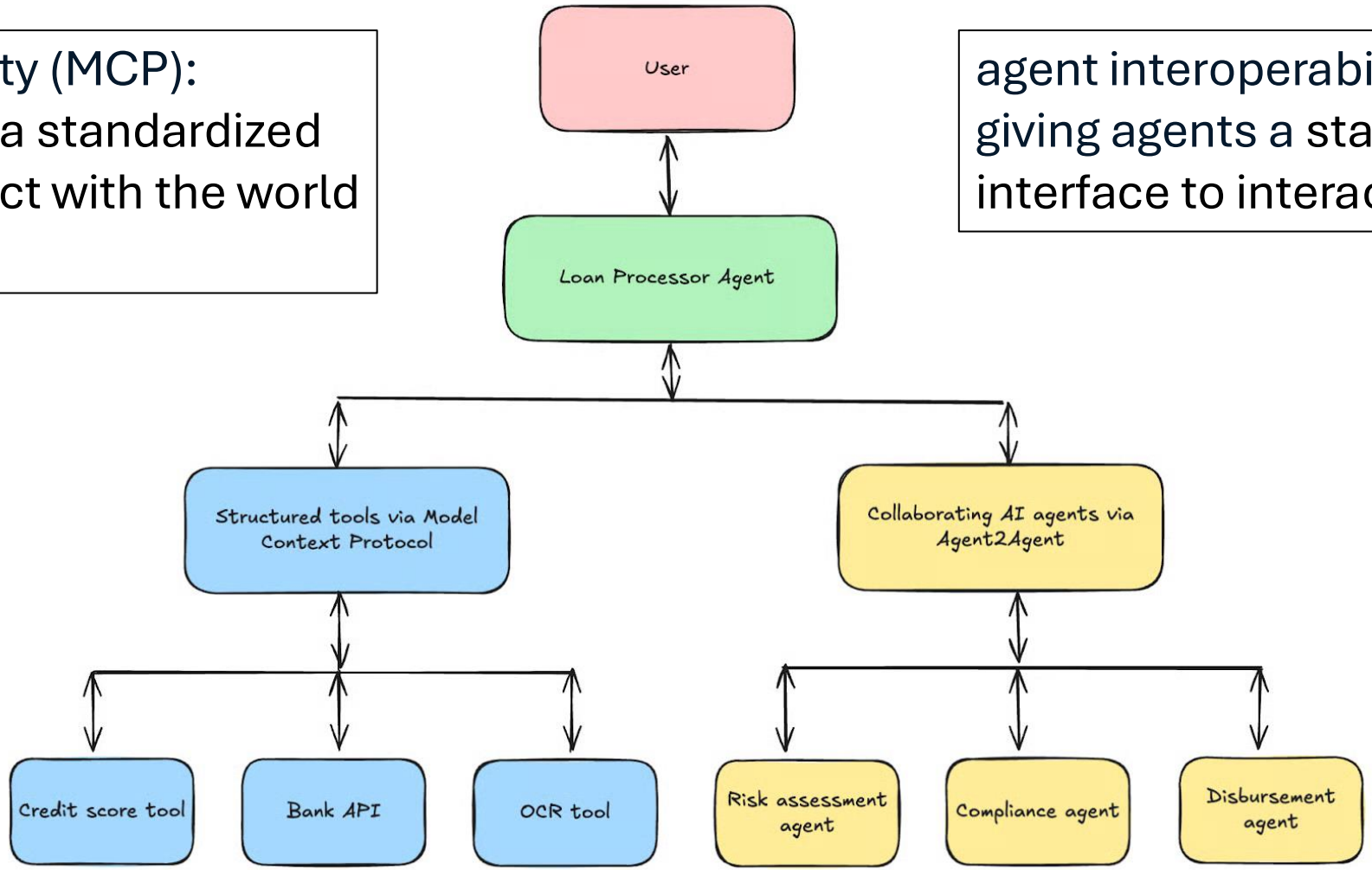
LLM orchestration + autonomy/planning loop

+ coordination between multiple agents

e.g., AutoGen

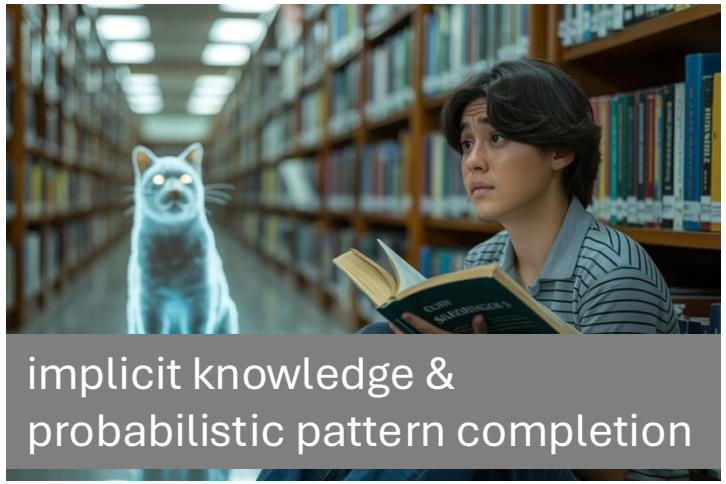
Example for Combination of A2A and MCP

tool interoperability (MCP):
giving each agent a standardized interface to interact with the world (tools, data, APIs)



agent interoperability (A2A):
giving agents a standardized interface to interact with each other

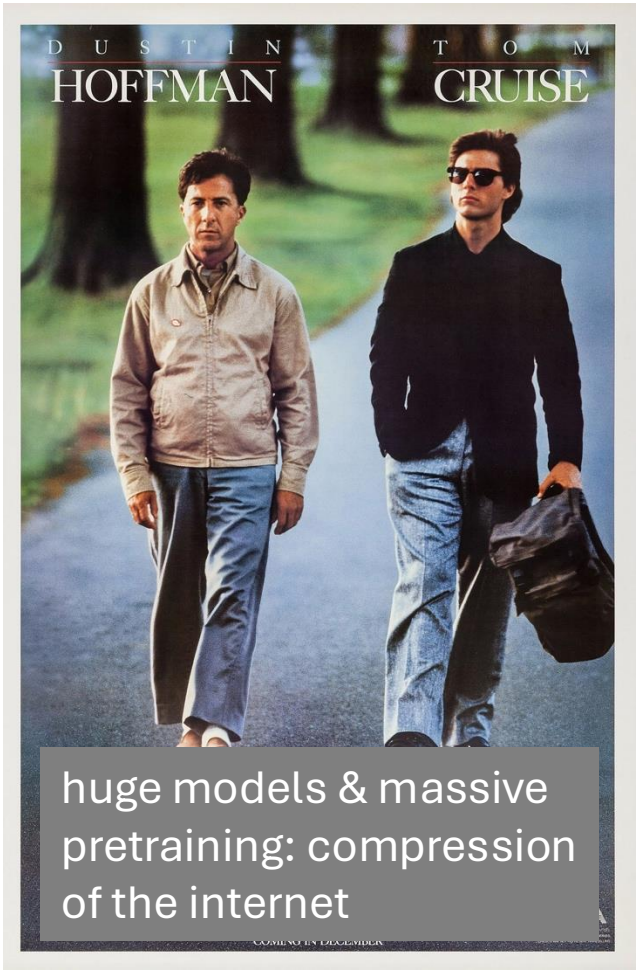
hallucinations



implicit knowledge & probabilistic pattern completion

LLM: *kind of a lossy simulation of a savant with cognitive issues*

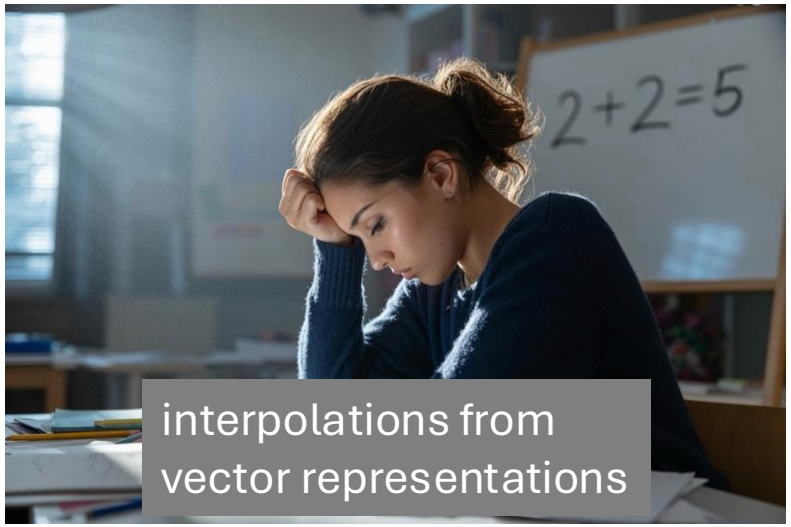
encyclopedic knowledge



huge models & massive pretraining: compression of the internet

only short-term memory in context

anterograde amnesia



interpolations from vector representations

jagged intelligence

from Karpathy

Software 1.0

computer code

↓ programs

computer

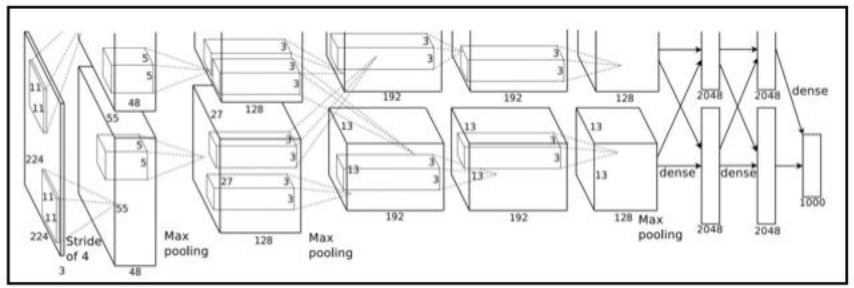
became programmable in **~1940s**

Software 2.0

weights

↓ programs

neural net



fixed function neural net

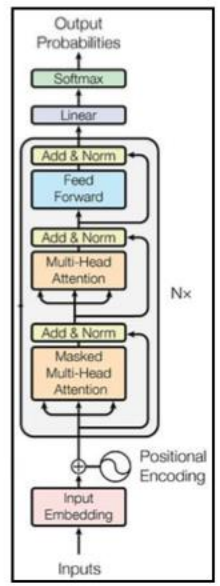
e.g. AlexNet: for image recognition (**~2012**)

Software 3.0

prompts

↓ programs

LLM



~2019

LLM = programmable neural net!

source

Example: Sentiment Classification

Software 1.0

```python
def simple_sentiment(review: str) -> str:
    """Return 'positive' or 'negative' based on a tiny keyword lexicon."""
    positive = {
        "good", "great", "excellent", "amazing", "wonderful", "fantastic",
        "awesome", "loved", "love", "like", "enjoyed", "superb", "delightful"
    }
    negative = {
        "bad", "terrible", "awful", "poor", "boring", "hate", "hated",
        "dislike", "worst", "dull", "disappointing", "mediocre"
    }

    score = 0
    for word in review.lower().split():
        w = word.strip(".,!?;:")          # crude token clean-up
        if w in positive:
            score += 1
        elif w in negative:
            score -= 1

    return "positive" if score >= 0 else "negative"
```

Software 2.0

10,000 positive examples

10,000 negative examples

encoding (e.g. bag of words)

train binary classifier

parameters

Software 3.0

You are a sentiment classifier. For every review that appears between the tags
<REVIEW> ... </REVIEW>, respond with **exactly one word**, either POSITIVE or NEGATIVE (all-caps, no punctuation, no extra text).

Example 1
<REVIEW>I absolutely loved this film—the characters were engaging and the ending was perfect.</REVIEW>
POSITIVE

Example 2
<REVIEW>The plot was incoherent and the acting felt forced; I regret watching it.</REVIEW>
NEGATIVE

Example 3
<REVIEW>An energetic soundtrack and solid visuals almost save it, but the story drags and the jokes fall flat.</REVIEW>
NEGATIVE

Now classify the next review.

source

LLMs ≠ AGI

current AI good at learning statistical patterns and making predictions

but no real “understanding” or world model

And don’t get fooled by alignment ;).