Generative AI and Symbolic Knowledge Representations
LLMs, Knowledge and Reasoning
1

Damir Cavar & Billy Dickson
ESSLLI 2024

July 2024

Course Page

- https://damir.cavar.me/ESSLLI24_LLM_KG.github.io/

Resources and Contact

- Website
- GitHub repo 1
- Damir's Python notebooks on GitHub

- Get in touch: @iu.edu (during ESSLLI 2024 also in person)
  - Damir Cavar: dcavar
  - Billy Dickson: dicksonb

Questions

- Who has never used Python and Jupyter Notebooks?
- Who has never experimented with Word Embeddings?
- Who has no background in Computational Linguistics or Natural Language Processing?
- Who has some understanding of how Large Language Models work?

Questions

- Who does not have experience with Knowledge Graphs?
- Who does not have experience with Description Logic, OWL, RDF, and Triple Stores?
- Who has never interacted with ChatGPT, Claude 3, Gemini, Llama 3, etc.?

Section 1

- Vector Models
- Words, Text, Semantic Space

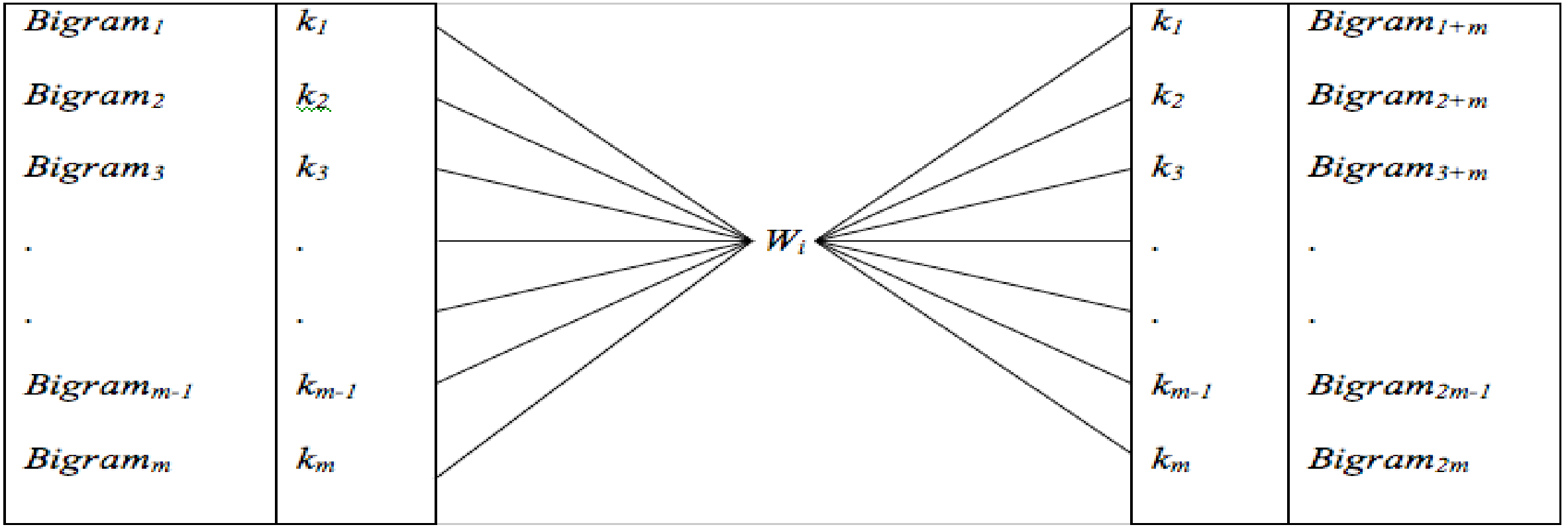
Words and Documents

- Document Vectorization (Jurafsky & Martin, slp3)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Lexical Vector Space



Vector Space Model of Distributional Properties

- Count co-occurrence of words in a bag of words 5 words left and right:

	…	cheese	bark	purr	fly	swim	…
dog	…						…
cat	…						…
bird	…						…
mouse	…						…
…	…	…	…	…	…	…	…

Vector Space

Left and right context:

			left					right		
	…	the	a	big	…	…	the	a	big	…
dog	…	45	32	3	…	…	0	0	0	…
cat	…	39	27	1	…	…	0	0	0	…
walked	…	0	0	0	…	…	67	49	1	…

Vector Models

- Why vector models?
  - Architecture argument:
    - Neural Models for NLP have a fixed input-output architecture: vectors of a specific length
    - Mapping language units to mathematical representations that can be efficiently computed
- What kind of vector models?
  - Hand-crafted feature vectors
  - Distributional vectors representing words in context (also tf-idf)
  - Machine-learned distributional semantics dense vectors
    - Encoding semantic similarity and relatedness reflected in distance in semantic space

Distributional Semantics

- Distributional Properties correlate with semantic similarities

- Word similarities:
  - Euclidean Distance

$$d = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

  - Cosine Similarity

$$S_C(A, B) := cos(\theta) = \frac{A \cdot B}{|A|\,|B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

cosine similarity $= S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$

Vector Models

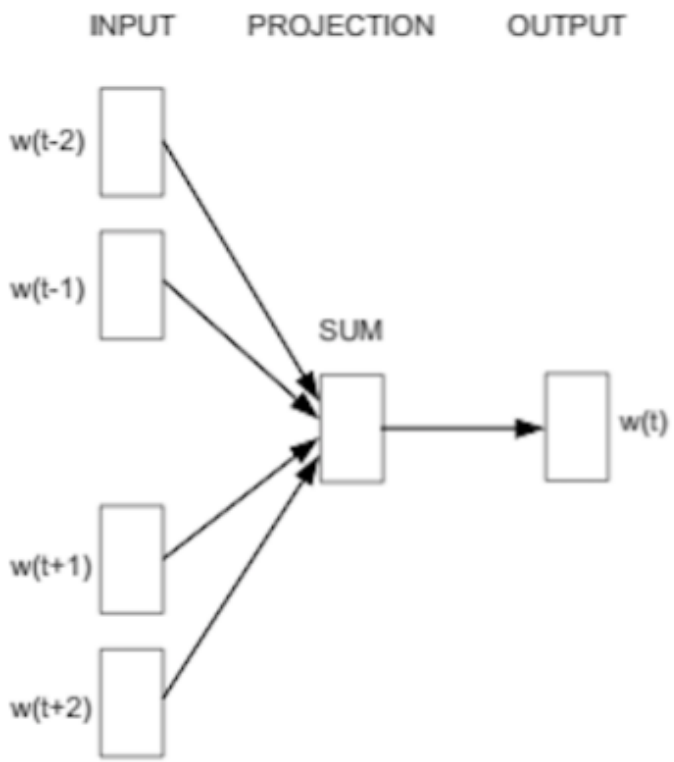
- Word (and text) embeddings as n-dimensional real vectors in classical Natural Language Processing (NLP)/AI applications
  - Word2vec (Mikolov et al., 2013)
  - fastText (Bojanowski et al., 2017; Joulin et al., 2018)
  - GloVe (Pennington et al. 2014)
  - Numberbatch (Speer et al., 2017)
  - BERT (Devlin et al., 2019)
    - Many derivatives

Training Vector Models

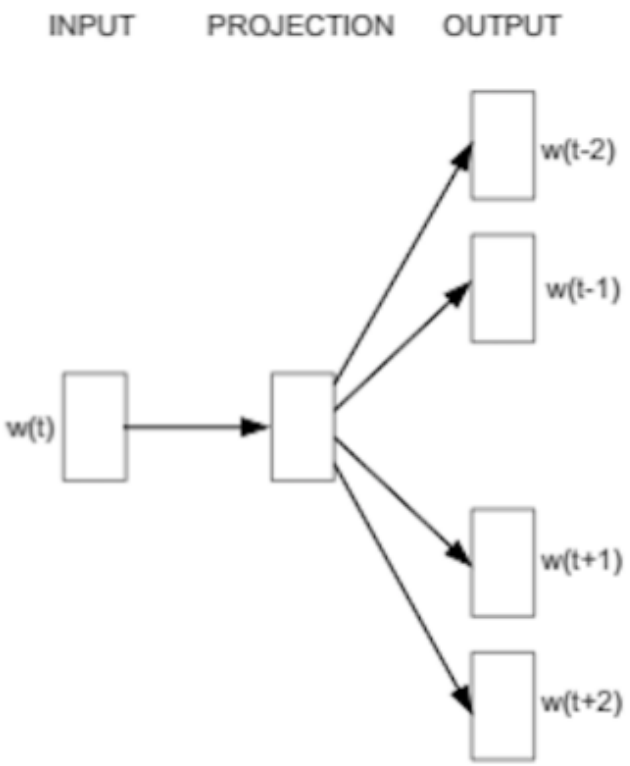
- **Continuous bag-of-words model**: This model predicts the middle word based on surrounding context words, i.e., *n* words before and after the current (middle) word, and the order of words in the context is not important.
- **Continuous skip-gram model**: predicts words within a certain range before and after the current word in the same sentence.

- **Predicting: (see tutorial)**
  - Using a neural model, word vectors are rows in a hidden layer.

Word2vec Predictions

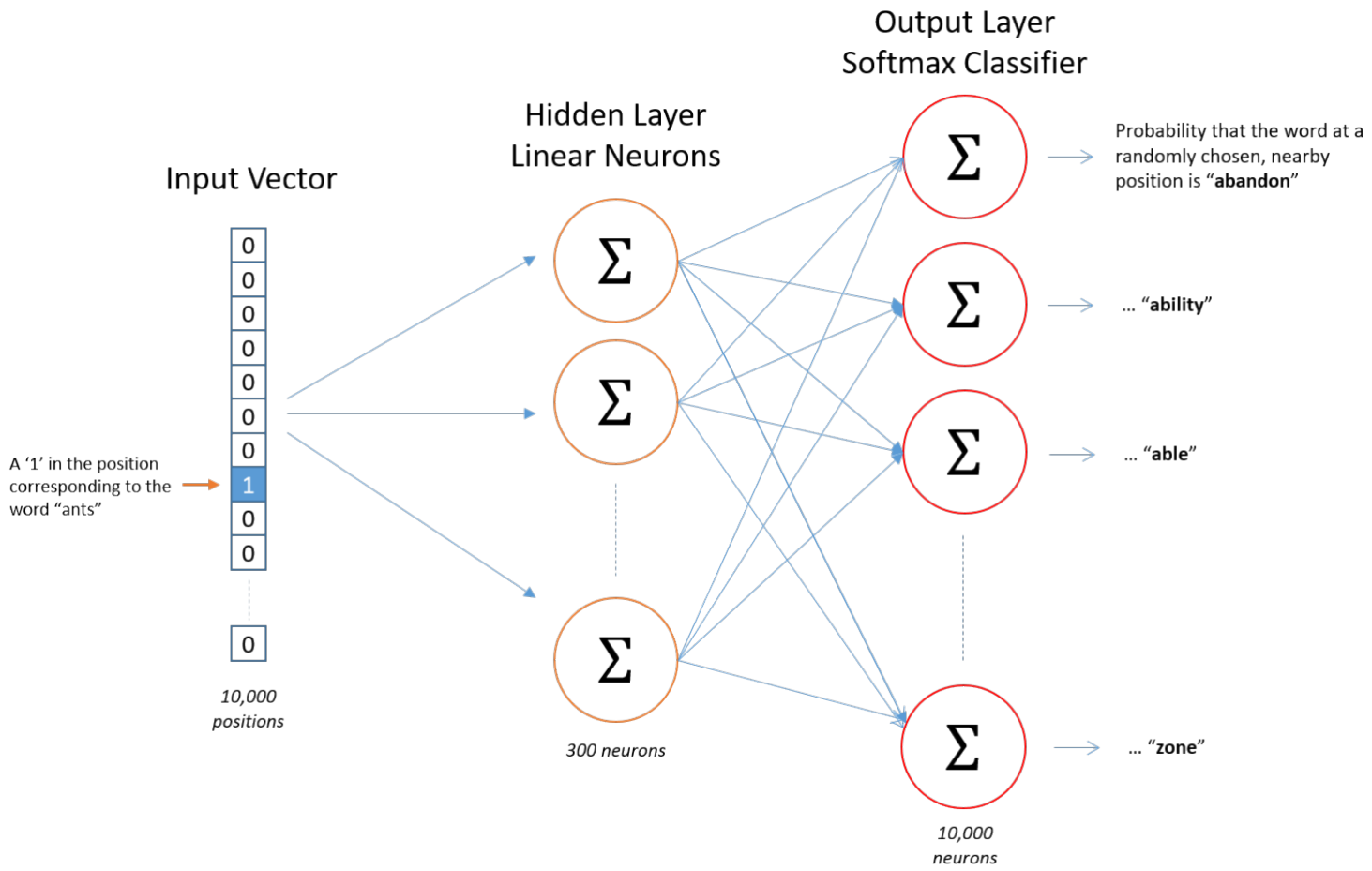


CBOW

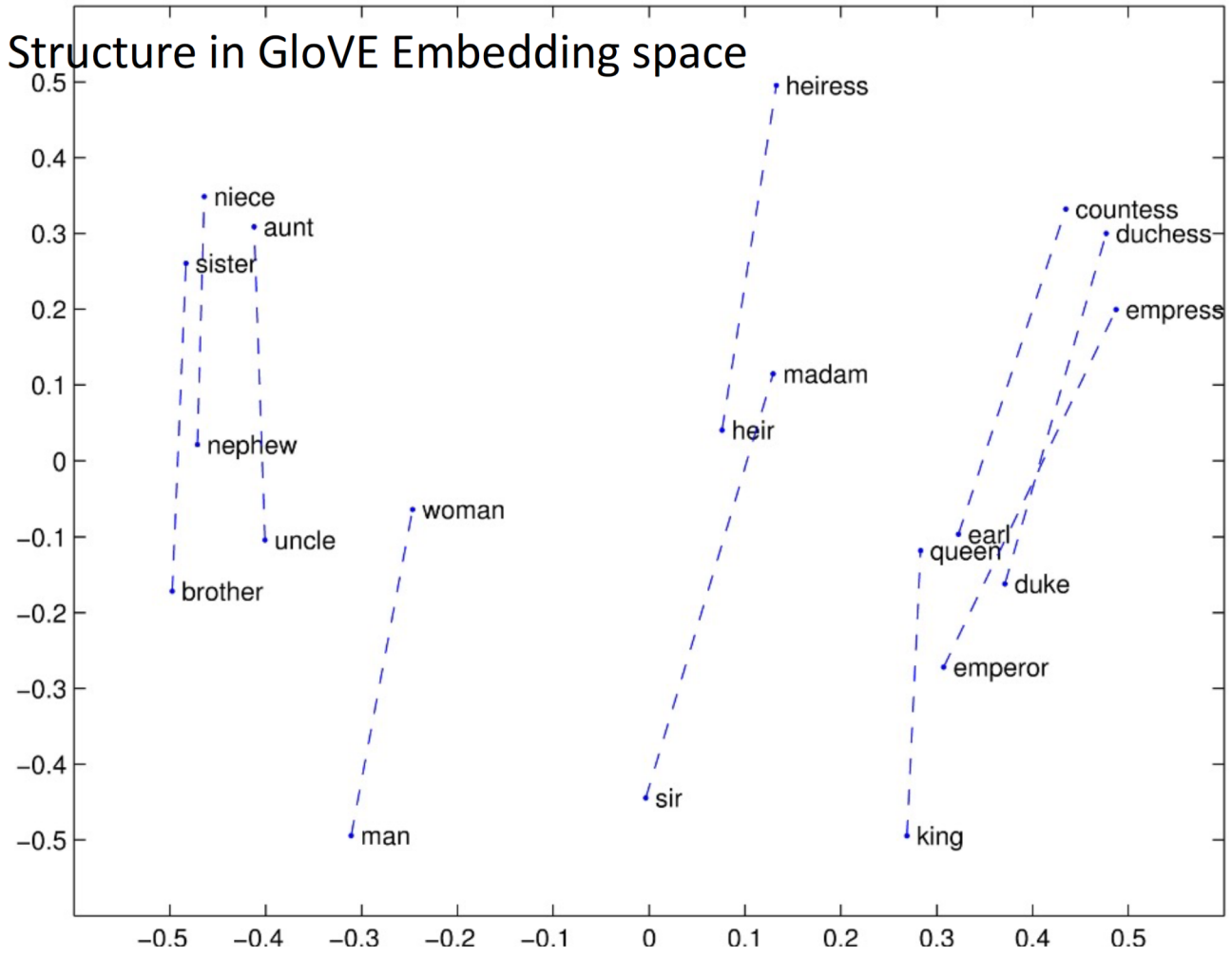


Skip-gram

Word2vec as a Network



Similarity


Structure in GloVE Embedding space
heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5

Vector Models

- Issues with machine-learned vector models?
  - Large amount of data necessary
  - Costly: Time, Computational Effort (and carbon emission)

- Advantages:
  - Many models are freely and openly available online
    - GitHub, Hugging Face, NLP-pipeline providers

Training Vector Models

- **Cloze test:**
  1. Used in Machine Learning – Marked Word Prediction in BERT (LM)
     - *The house ___ I was born.* (a. *where* , b. *which*)
  2. Next word prediction as in Large Language Models (LLMs)

- Tokens = Words or word fragments in BERT
- Tokens ≠ Words in LLMs
  - Byte-pair encoding

Vector Models

- Word embeddings vs. Distributional Vector Models
  - fastText: 300-dimensional word vectors, 2.5 mil. Words
  - GloVe: 840 billion tokens, 300-dimensional word vectors, 2.1 mil. Words
  - Numberbatch: 300-dimensional vectors, 516,783 words
  - BERT: 768-dimensional word vectors
- Dimensions:
  - Real numbers
    - Example: 300 x real = 300 x 32 bit = 9,600 bit
    - Real numbers could also be encoded using 64 bit

Vector Models

- Word2Vec
  - Lexical ambiguity is not captured, i.e., *date* (fruit, time expression)
  - Limited vocabulary (retraining for new words)
    - Unknown Word Problem
- fastText
  - Solves the Unknown Word Problem by using sub-word segments (not true morphemes)
- Numberbatch
  - Merges previous models and adds ConceptNet Knowledge Graph representations to improve semantic properties
    - Graph Embeddings (Concepts instead of strings)

Vector Models

- BERT
  - Adds sub-word tokens: WordPiece algorithm
    - *locking* could be tokenized as *lock* and *##ing*
    - *##ing* indicates that *ing* should be treated like a word continuation, i.e., a dependent morpheme in the linguistic sense
  - Processing of sequences of words
  - reduces the unknown word problem and increases the coverage at the lexical level
  - WordPiece in Schuster and Nakajima (2012) as a method to cope with out-of-vocabulary words and improve the general performance of machine learning and NLP tasks

Vector Models

- Orthographic / Linguistic tokenization
  - Too large a number of tokens (= number of embeddings)
- WordPiece and similar strategies:
  - Still a large number of tokens on multi-lingual content, programming code, data structures, etc.
- Simpler encoding and tokenization:
  - Byte-pair encoding
    - non-linguistic or orthographic
    - language independent
  - Predefined vocabulary size

Tokenization

- Tokenization methods:
  - Word tokenization
  - WordPiece
  - Byte-Pair Encoding (BPE)
  - Unigram
  - SentencePiece

Word Tokenization

- Training data might be missing words: the Unknown Word Problem
  - The training missing: football
  - But not missing: "foot" or "ball"
- Inflected forms in specific tense, case, number, gender agreement might have been never seen

- A lemmatizer might help, but: the concrete lemmatized token might be out of vocabulary

Word Tokenization

- Other problems:
  - slang
  - abbreviations

- Or language specific orthography:
  - no spaces between words (e.g., Japanese, Chinese, Arabic)

Tokenization

- Alternative: Character-based tokenization

- Compute embeddings for characters
- Compose words as embeddings using character embeddings

- Robust with input processing
  - Typos
  - Variation in orthography
  - Unknown words

Character Tokenization

- Problem:
  - each character is a token
  - this results in a lot more tokens
  - this results in a lot more input computation
- Example:
  - 5 word sentence might have 30 character tokens and 5 word tokens
- Word generation using characters can result in wrong spellings

Character Tokenization

- Distributional semantics might not work with such models
- Compare:
  - listen
  - silent

- We would need character embeddings and positional encoding to catch the difference between the two words.

Sub-word Tokenization

- Character-based models:
  - risk losing the semantic features of the word
- Word-based tokenization:
  - very large vocabulary needed to encompass all the possible variations of every word
- Goal: develop an algorithm that could
  - retain the semantic features of the token, that is information per token
  - tokenize without demanding a very large vocabulary with a finite set of words

Sub-word Tokenization

- Breaking down the words based on a set of prefixes and suffixes:
  - identify sub-words like: "##s", "##ing", "##ify", "un##"...
  - the position of the double hash denotes prefix and suffixes

- Tokenize: "unhappily"
  - "un##", "happ", "##ily"
- Result: few subwords and combinatorics to create words
- Compression: memory requirement and little effort required to create a large vocabulary

Sub-word Tokenization

- Problems:
  - some created sub-words might be very rare and occupy space
  - language specific rules and tokens:
    - for every language we need to define a different set of rules to create sub-words - every language has a different morphology
- Strategy:
  - most modern tokenizers have a training phase:
    - identification of recurring text in the input corpus and create new sub-word tokens
  - rare patterns: stick to word-based tokens

Sub-word Tokenization

- Important factor:
  - vital role takes the size of the vocabulary
  - large vocabulary size allows for more common words to be tokenized
  - smaller vocabulary requires more sub-words to be created to create every word in the text without using the <UNK> token

Byte-Pair Encoding

- Byte-Pair Encoding (BPE) used in language models:
  - GPT-2
  - RoBERTa
  - XLM
  - FlauBERT
  - ...

- Pre-tokenization:
  - space tokenization

Byte-Pair Encoding

- Simple form of data compression algorithm:
  - the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data

- first described in the article “A New Algorithm for Data Compression” published in 1994

Byte-Pair Encoding

- Use in NLP to find the best representation of text using the smallest number of tokens:
  - Add an identifier (</w>) at the end of each word to identify the end of a word and calculate the word frequency in the text.
  - Split the word into characters and then count character frequency.
  - From the character tokens, for a predefined number of iterations, count the frequency of the consecutive byte pairs and merge the most frequently occurring byte pairing.
  - Iterate until the iteration or token limit is reached.

Different Pre-tokenizers

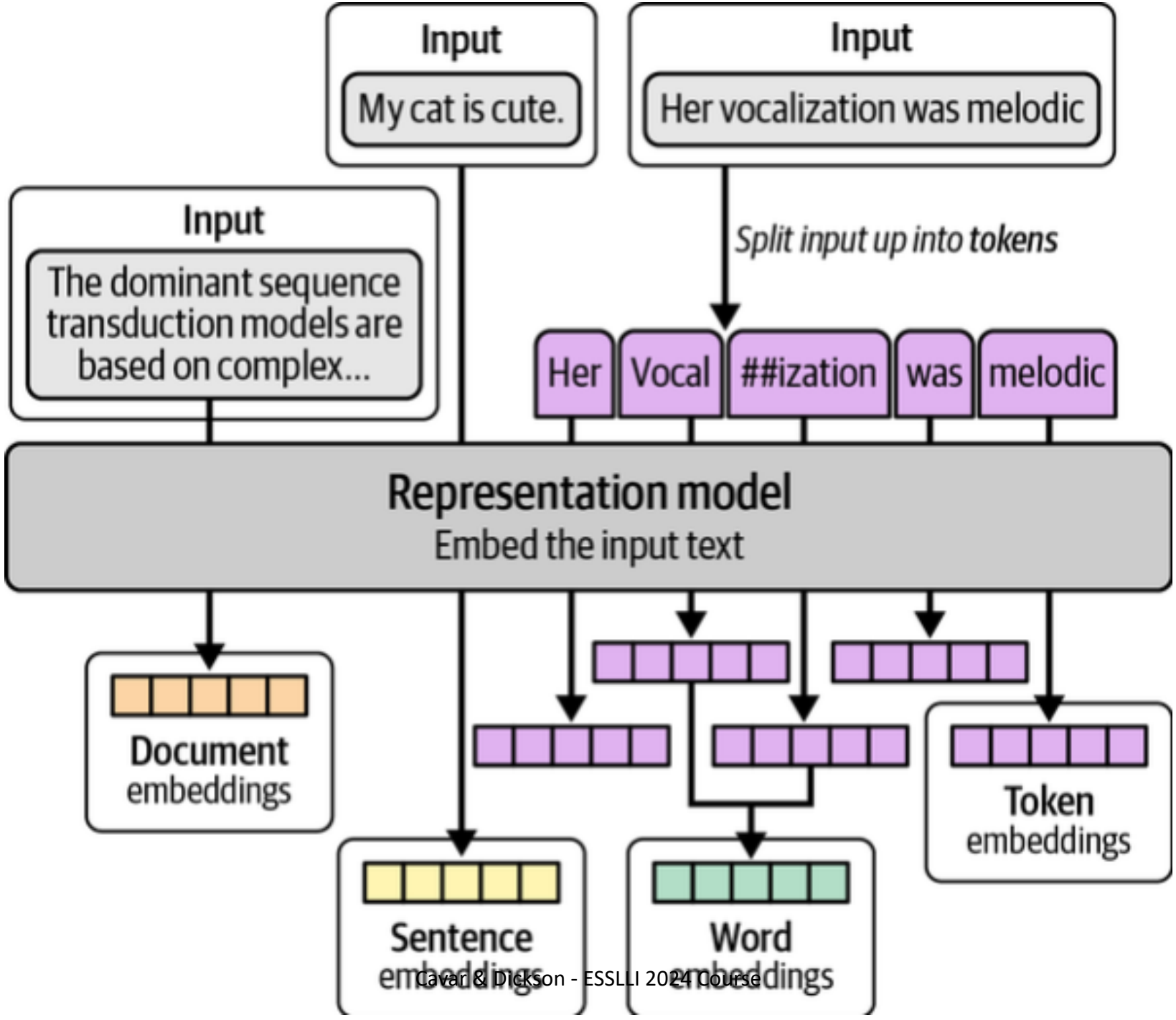
- Whitespace Pre-tokenizer:
  - "this", "sentence's", "content", "includes:", "characters,", "spaces,", "and", "punctuation.",
- BERT Pre-tokenizer:
  - "this", "sentence", "'", "s", "content", "includes", ":", "characters", ",", "spaces", ",", "and", "punctuation", ".",
- GPT-2 Pre-tokenizer:
  - "this", "Ġsentence", "'s", "Ġcontent", "Ġincludes", ":", "Ġcharacters", ",", "Ġspaces", ",", "Ġand", "Ġpunctuation", ".",
- ALBERT Pre-tokenizer:
  - "_this", "_sentence's", "_content", "_includes:", "_characters,", "_spaces,", "_and", "_punctuation.",

Next Word Prediction

- Applications
  - Texting
  - Chatbots
- Examples:
  - Finite State Machines
  - N-gram Models

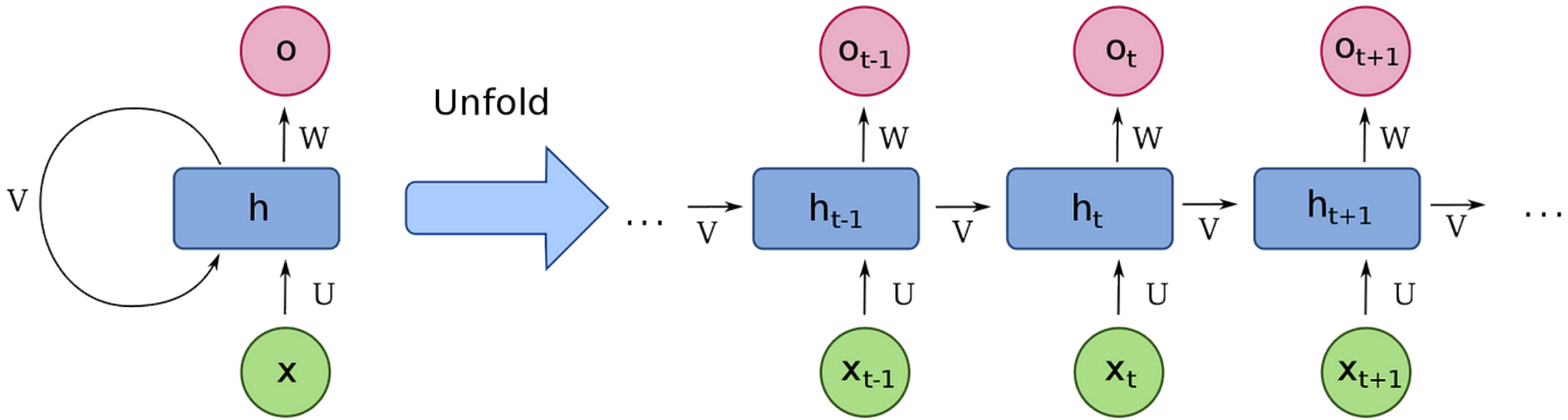
Section 2

- Recurrent Neural Networks
- Attention
- Transformers


Input
My cat is cute.
Input
Her vocalization was melodic
Input
The dominant sequence transduction models are based on complex...
Split input up into tokens
Her
Vocal
##ization
was
melodic
Representation model
Embed the input text
Document embeddings
Sentence embeddings
Word embeddings
Token embeddings

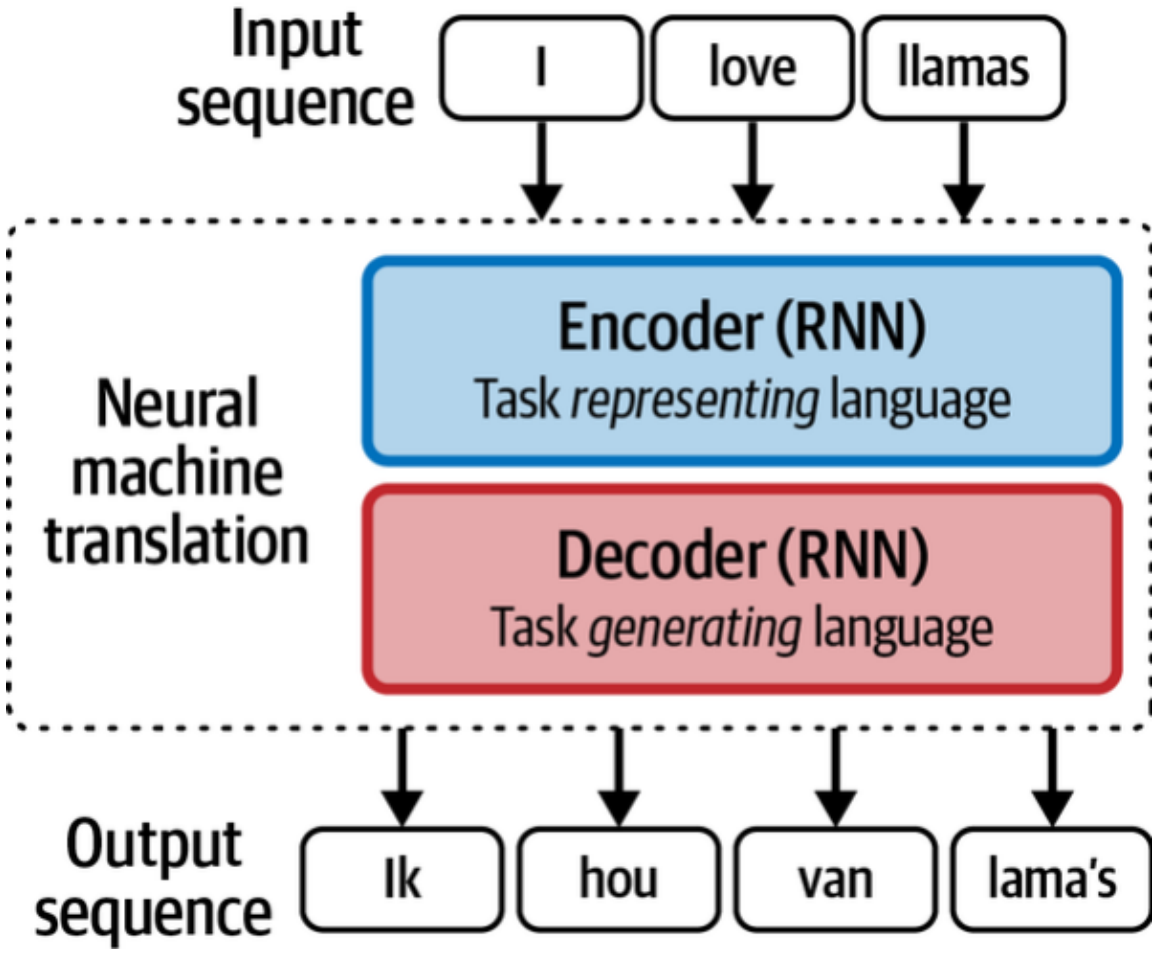
Recurrent Neural Networks (RNNs)

- **Sequence Data Processing:**
- RNNs are well-suited for tasks where the order of input data is important.
- They process inputs sequentially, maintaining a hidden state that captures information from previous steps.
- **Memory Capability:**
- RNNs can remember previous inputs due to their internal state, making them effective for tasks requiring context, like language translation or sentiment analysis.

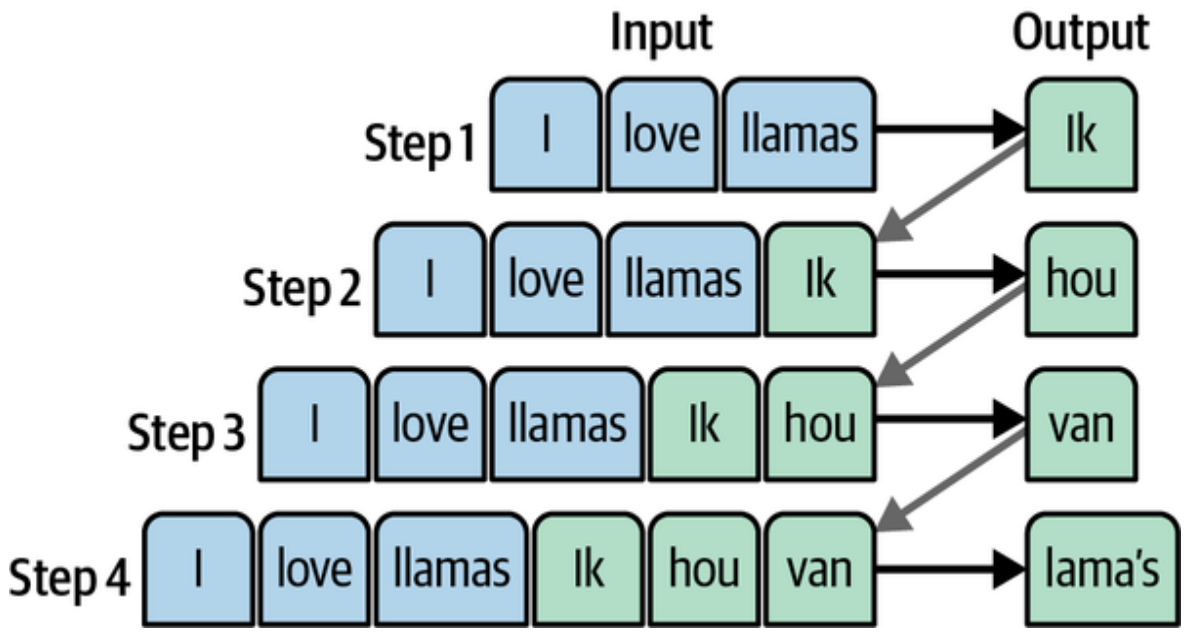

o
W
V
h
U
x
Unfold
o_{t-1}
o_t
o_{t+1}
h_{t-1}
h_t
h_{t+1}
x_{t-1}
x_t
x_{t+1}

Word2vec provides a static vector embedding for each word. We can use RNNs to encode the **context sensitive** word embeddings.

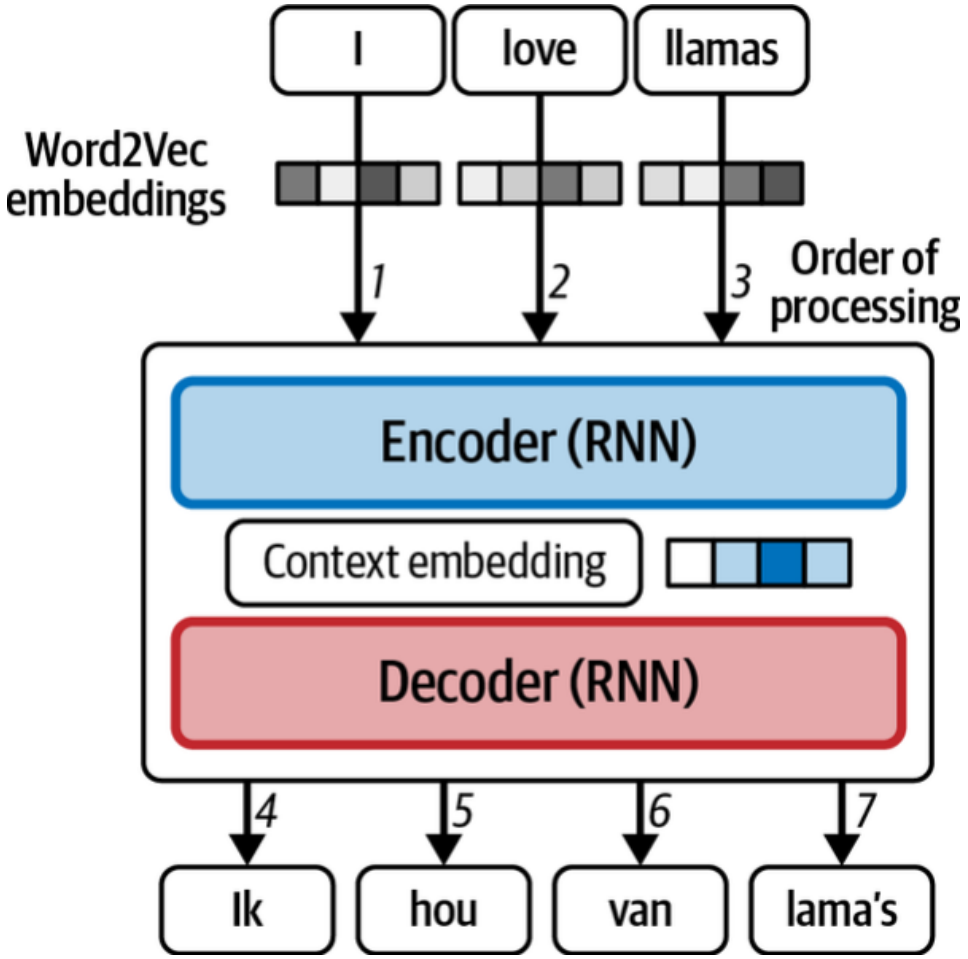
<eos> token signals that the model is finished generating



Word generation is **autoregressive:** when predicting the next word, the model must ingest all previously generated words



We can use **word2vec** as input the initial starting point. Note the order that each token is processed and generated.



Limitations of using the context embedding

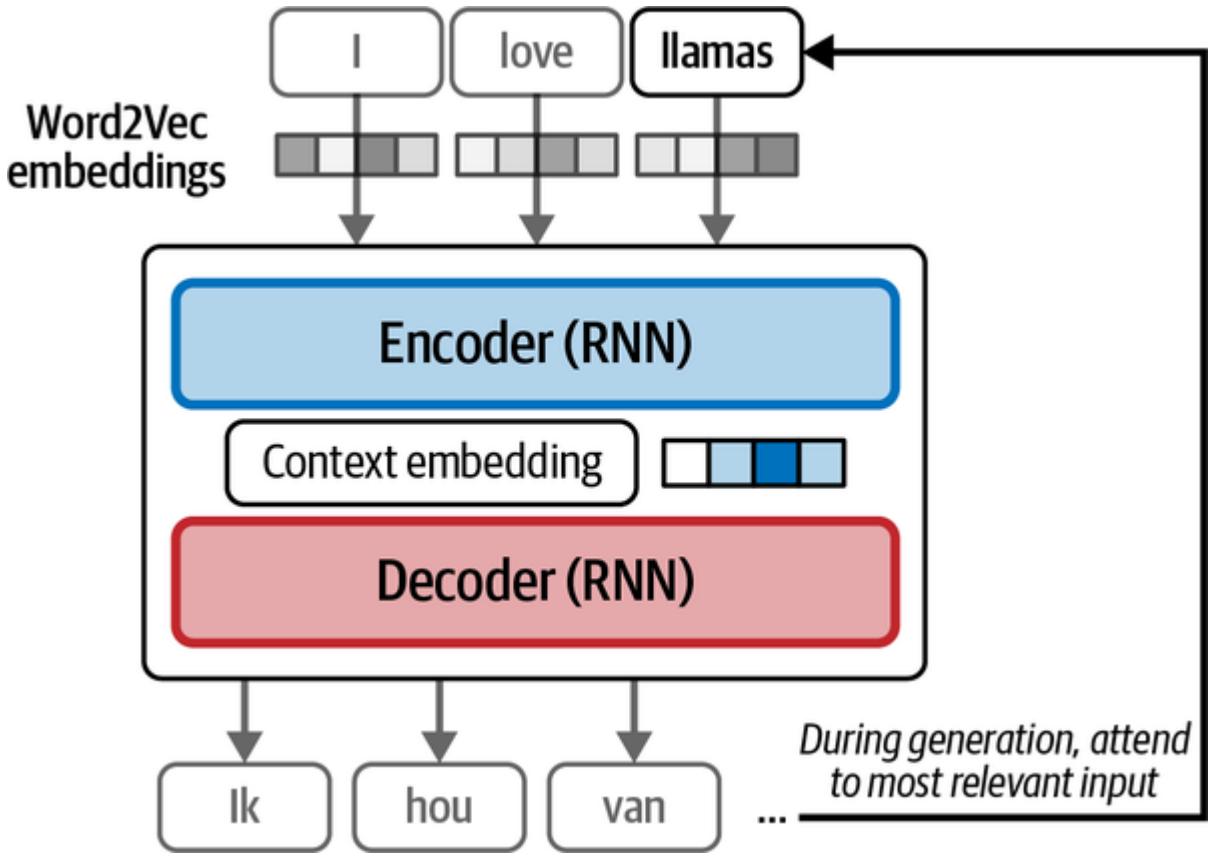
- **Information Bottleneck:**
  - Loss of important details due to compressing the entire sequence into a fixed-size vector.
- **Difficulty with Long Sequences:**
  - Struggles to retain information over long sequences, leading to the vanishing gradient problem.

Solution?

Solution?

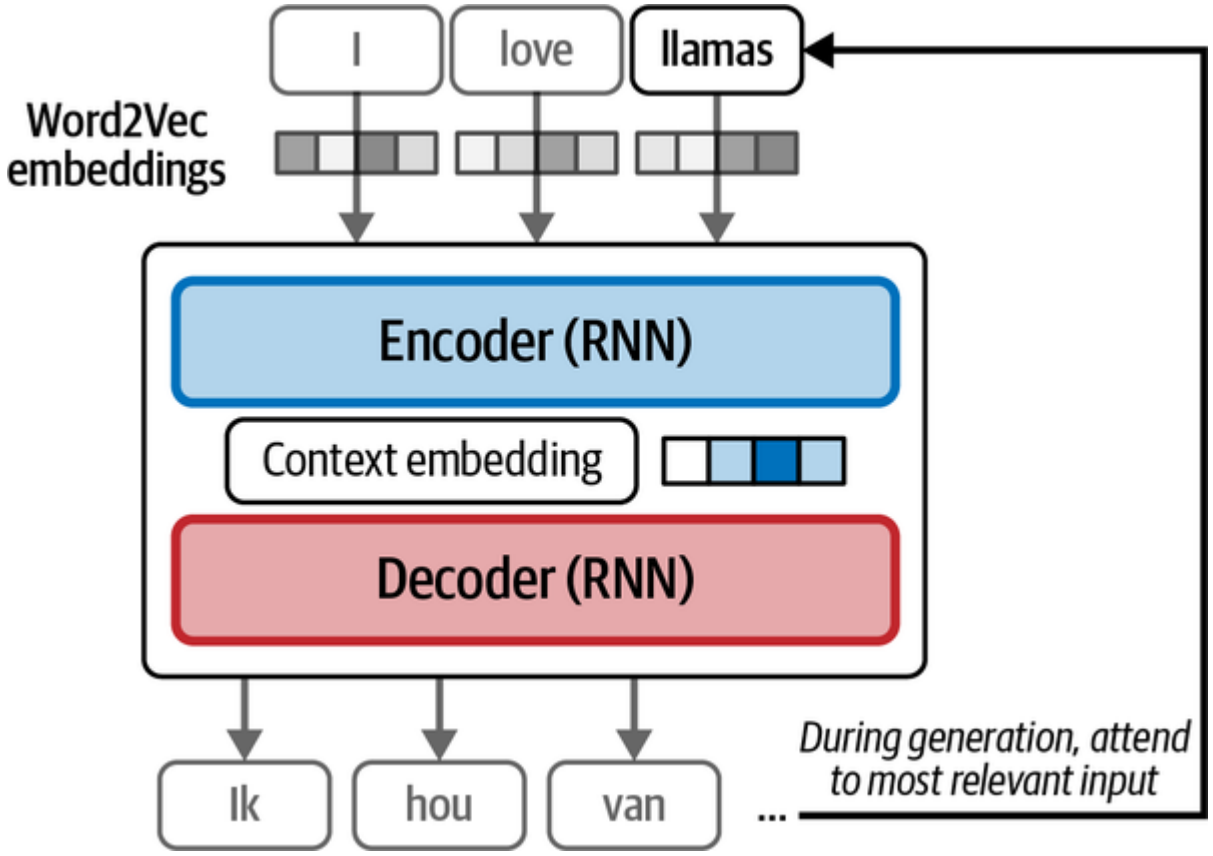
- Attention
  - The model pays attention to the most helpful part of the input for predicting the next word

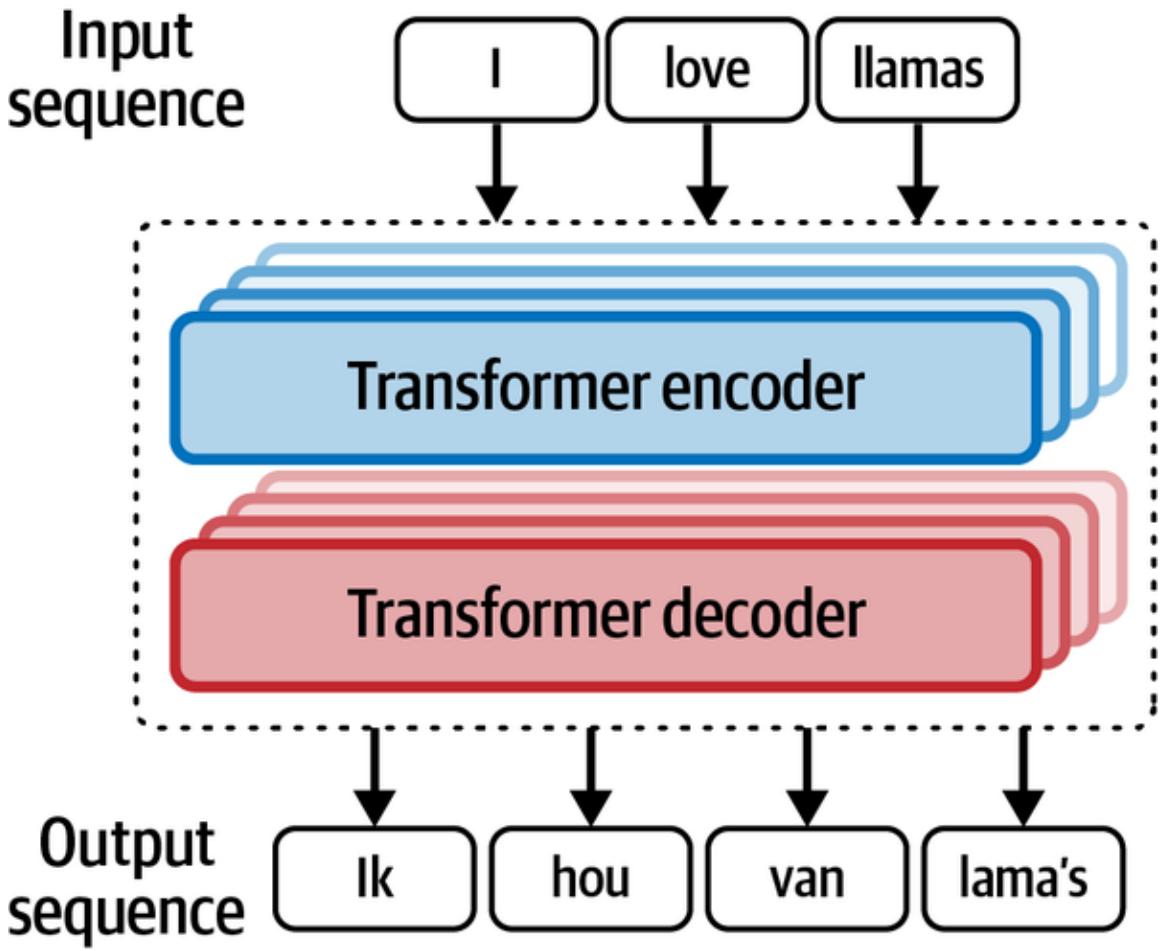
Model is trained to know how to distribute its attention



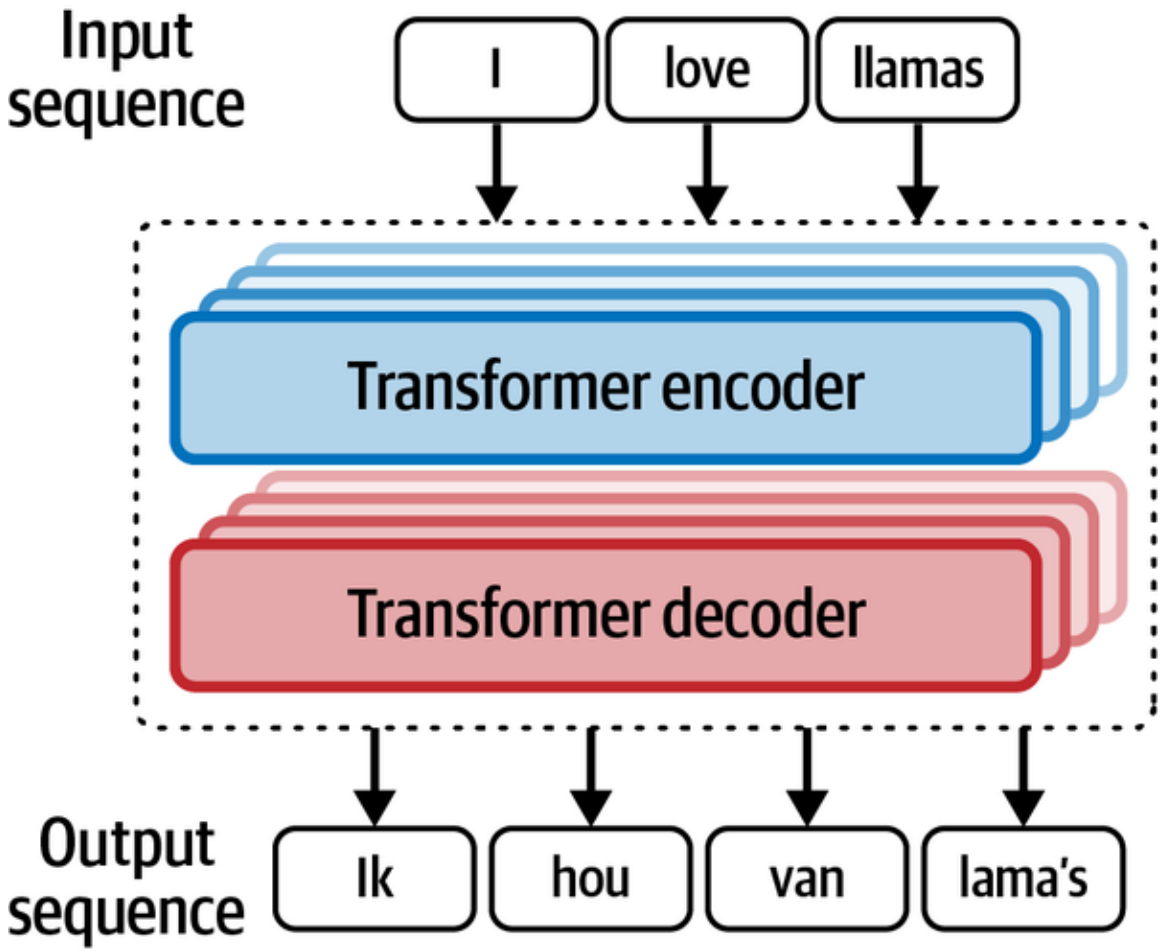
Model is trained to know how to distribute its attention

Still have problems with **catastrophic forgetting**



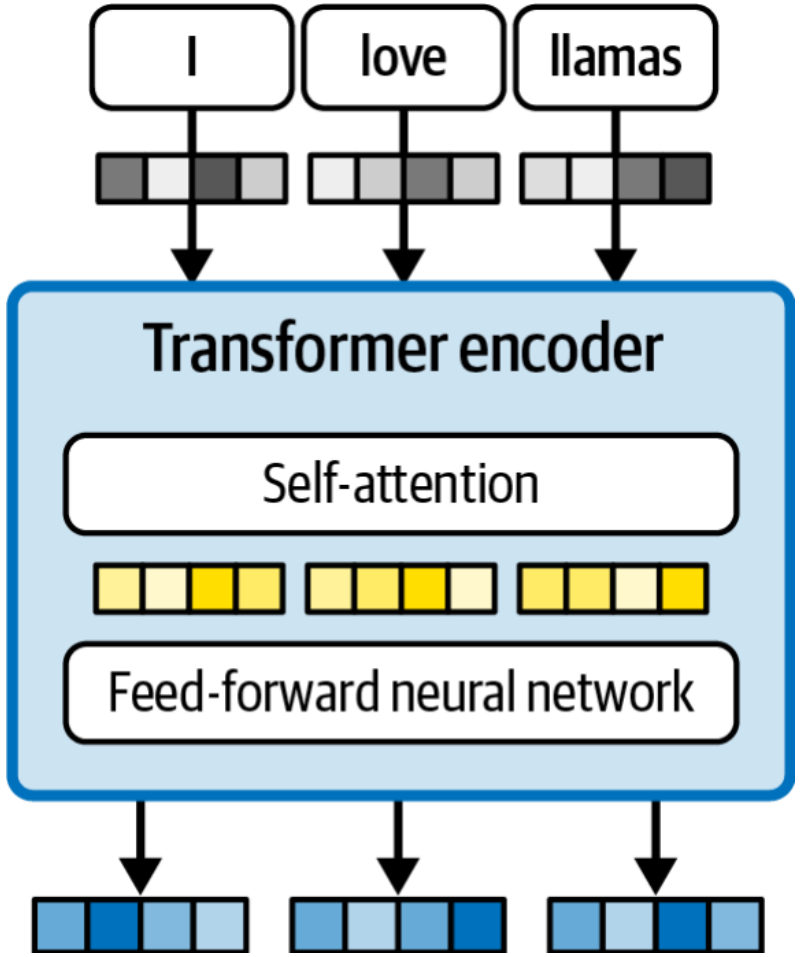

Input sequence
I
love
llamas
Transformer encoder
Transformer decoder
Output sequence
Ik
hou
van
lama's

Attention is All You Need
(Vaswani et al. 2017)

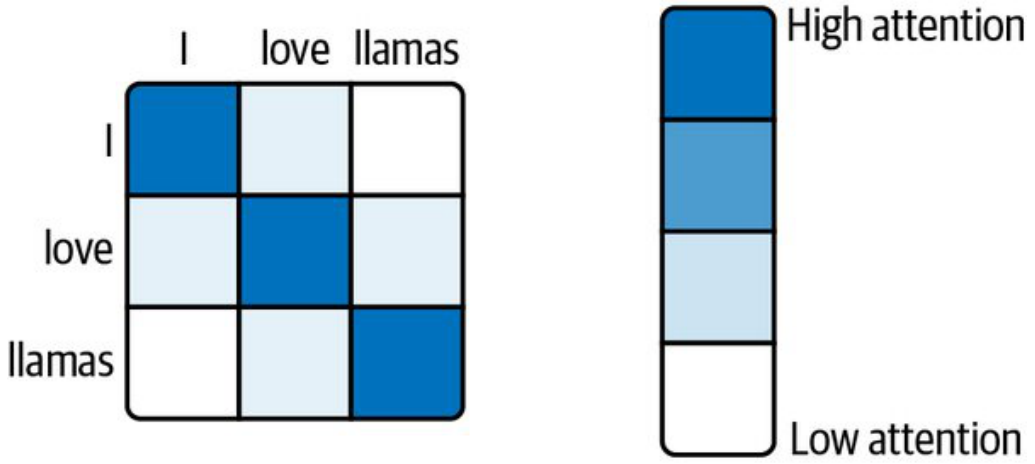


Input can be processed in parallel, all at once

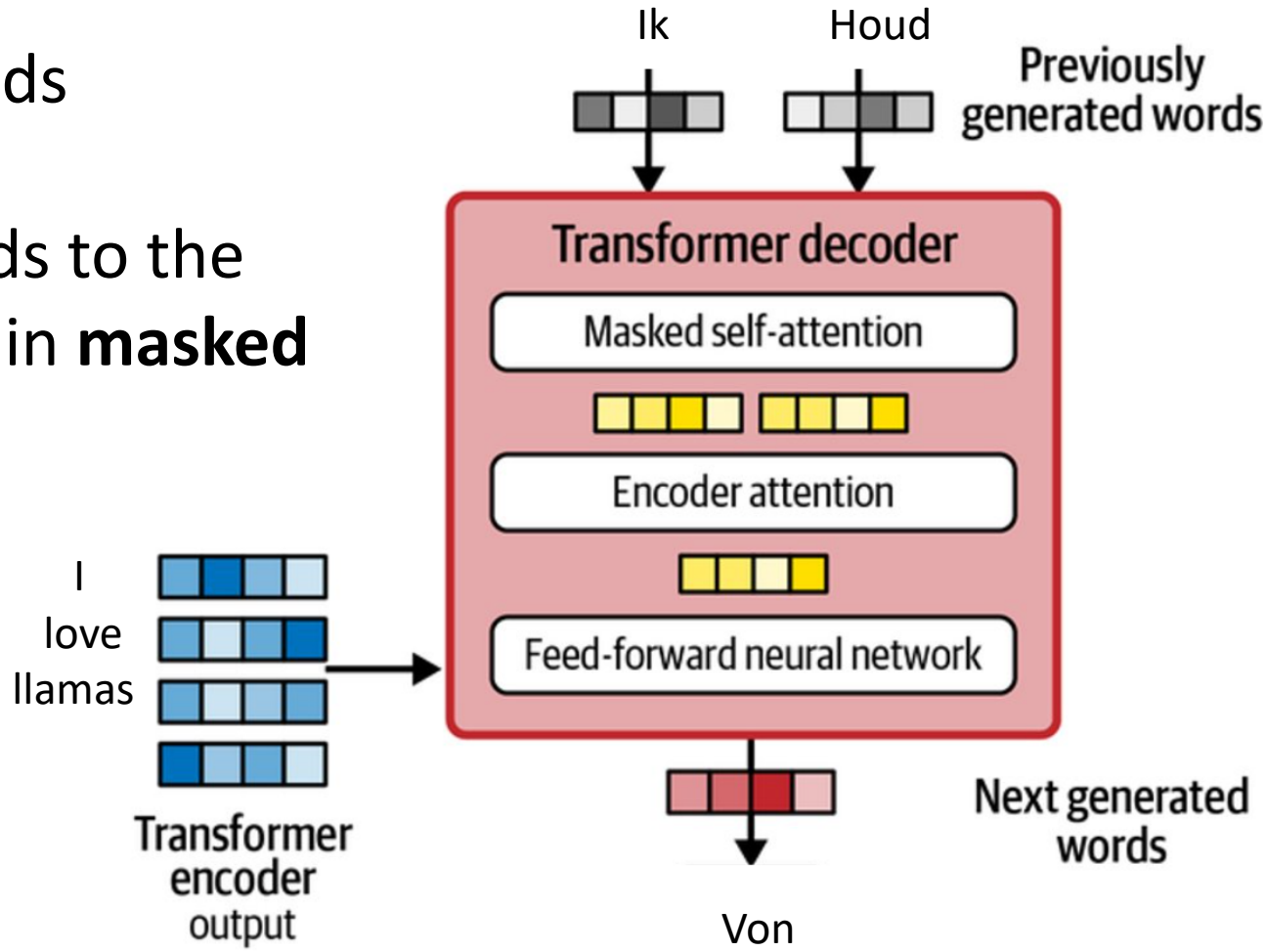
self-attention allows each token to distribute its attention to each other token including itself



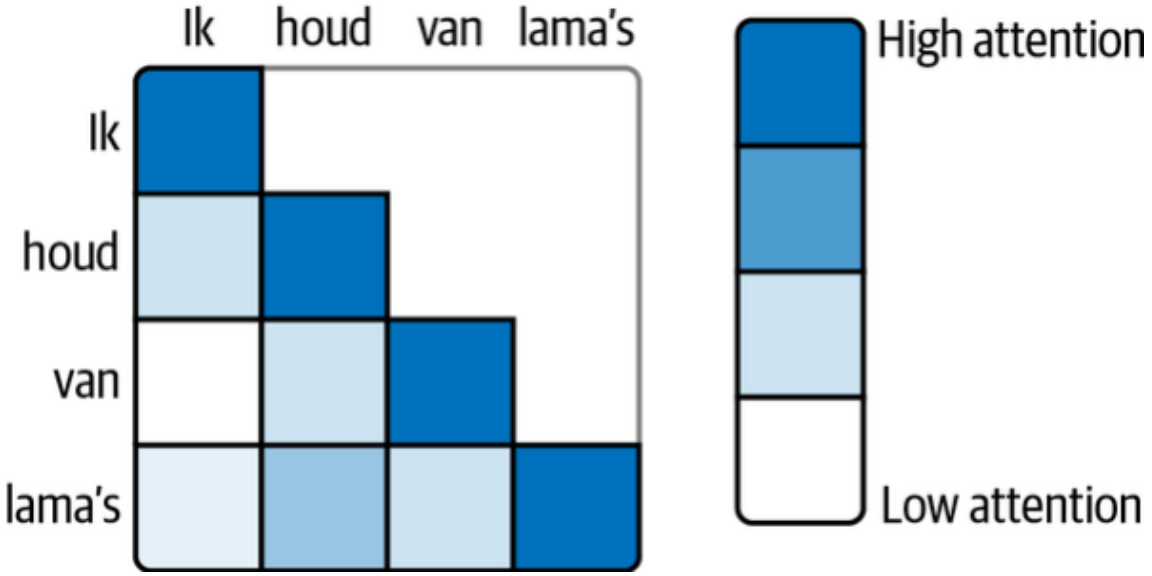
In the encoder, each token sees every other token of the input



In the decoder, the model attends to the output of the encoder in **encoder attention** and attends to the currently generated translation in **masked self-attention**



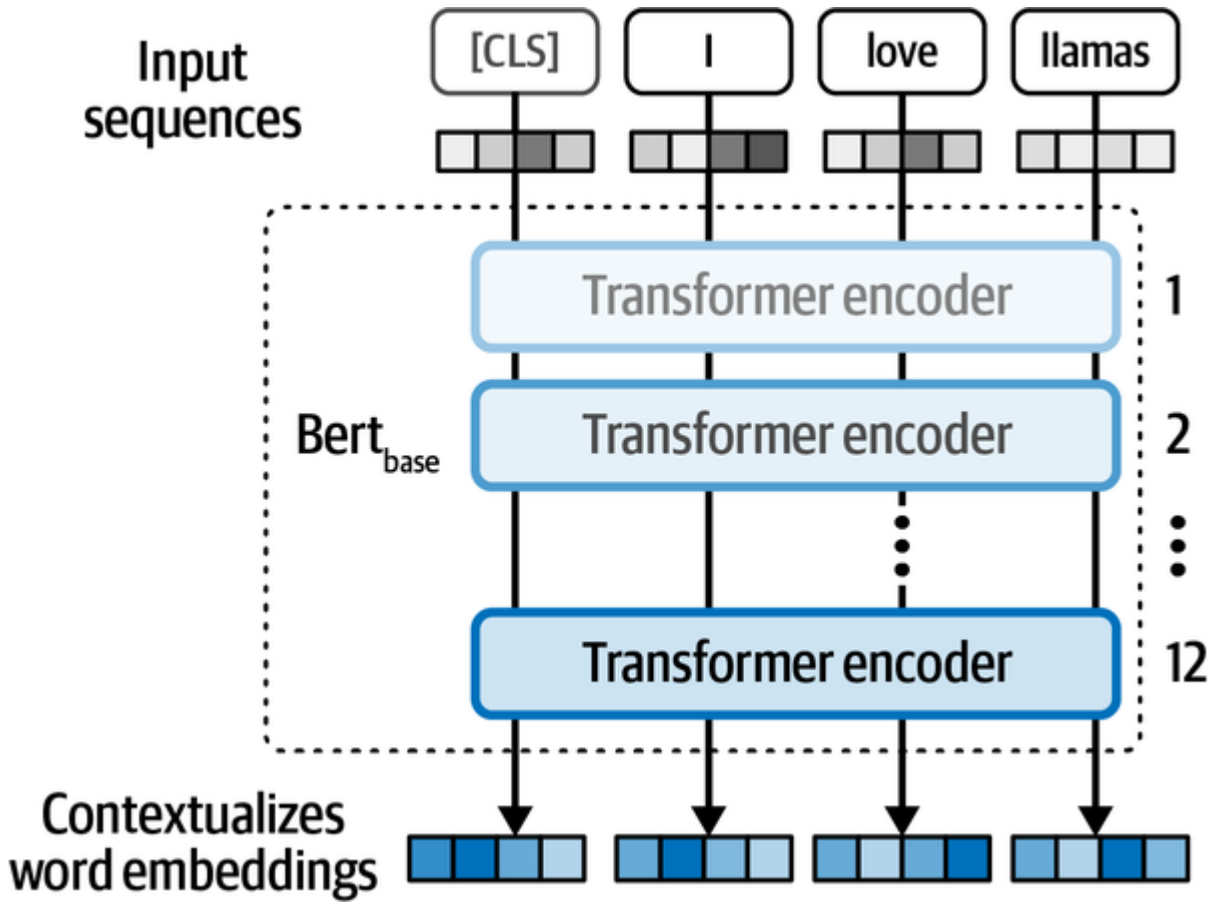
Masked self-attention prevents the model from cheating and looking into the future during training



Encoder Only Models

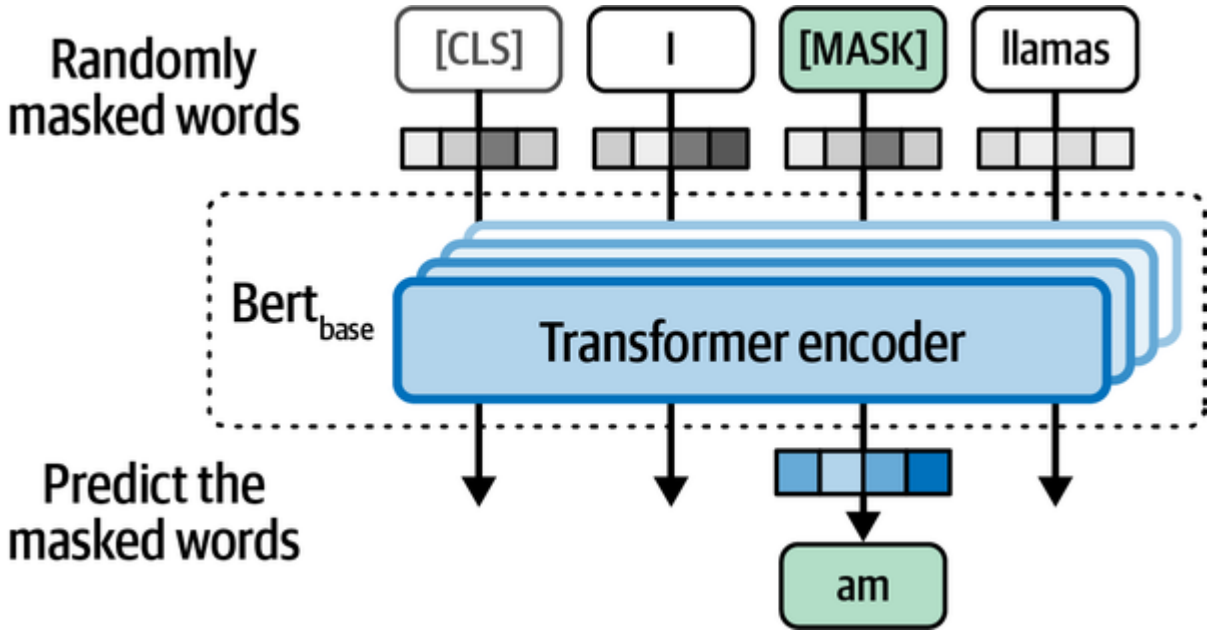
- **Purpose:**
  - Used for understanding and classification tasks.
  - Examples: Text classification, sentiment analysis, named entity recognition (NER).
- **Architecture:**
  - Multiple layers of encoders.
  - Processes input sequence to capture contextual information.
  - Outputs a fixed-size vector or sequence of vectors.
- **BERT (Bidirectional Encoder Representations from Transformers):**
  - Considers both past and future tokens for bidirectional context.

[CLS] or classification token, is used to represent the entire sequence

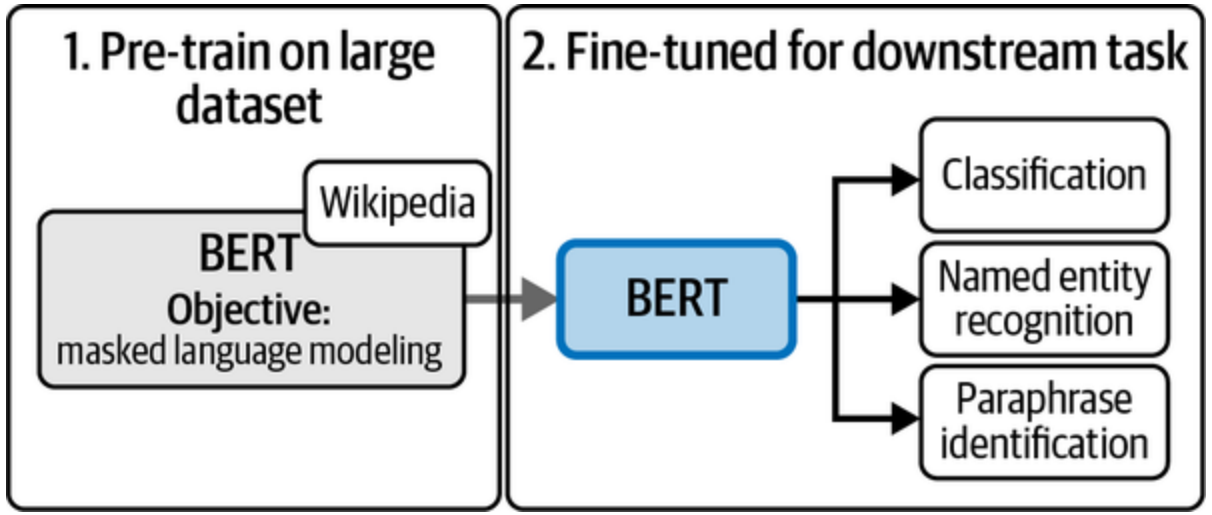


BERT implements **masked language modeling** which randomly blocks out tokens in the sequence.

The models must learn to fill in the missing gaps.



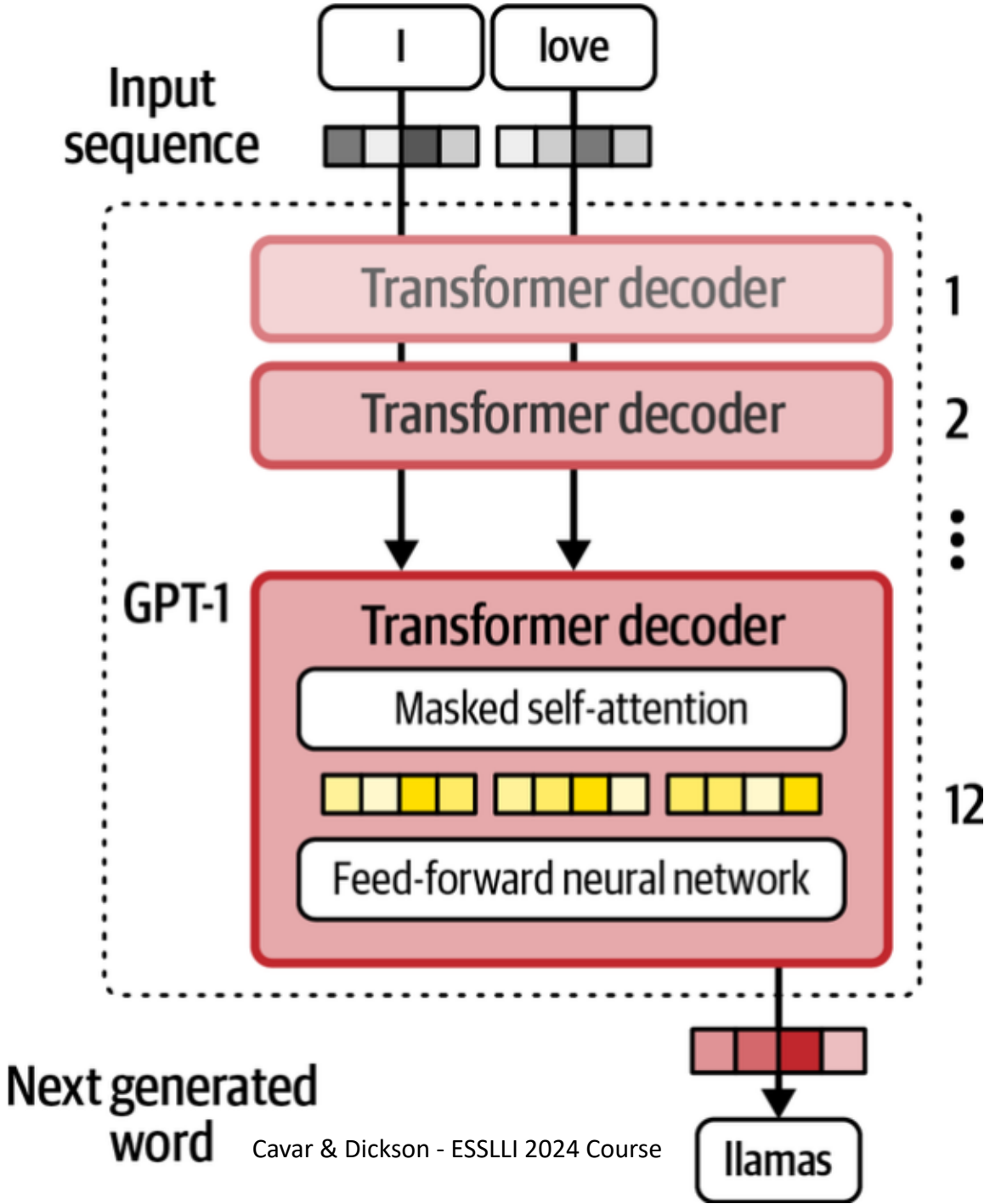
Fine-tuning involves initially pre-training on a large dataset, and then additionally training for a short amount of time on task specific data.

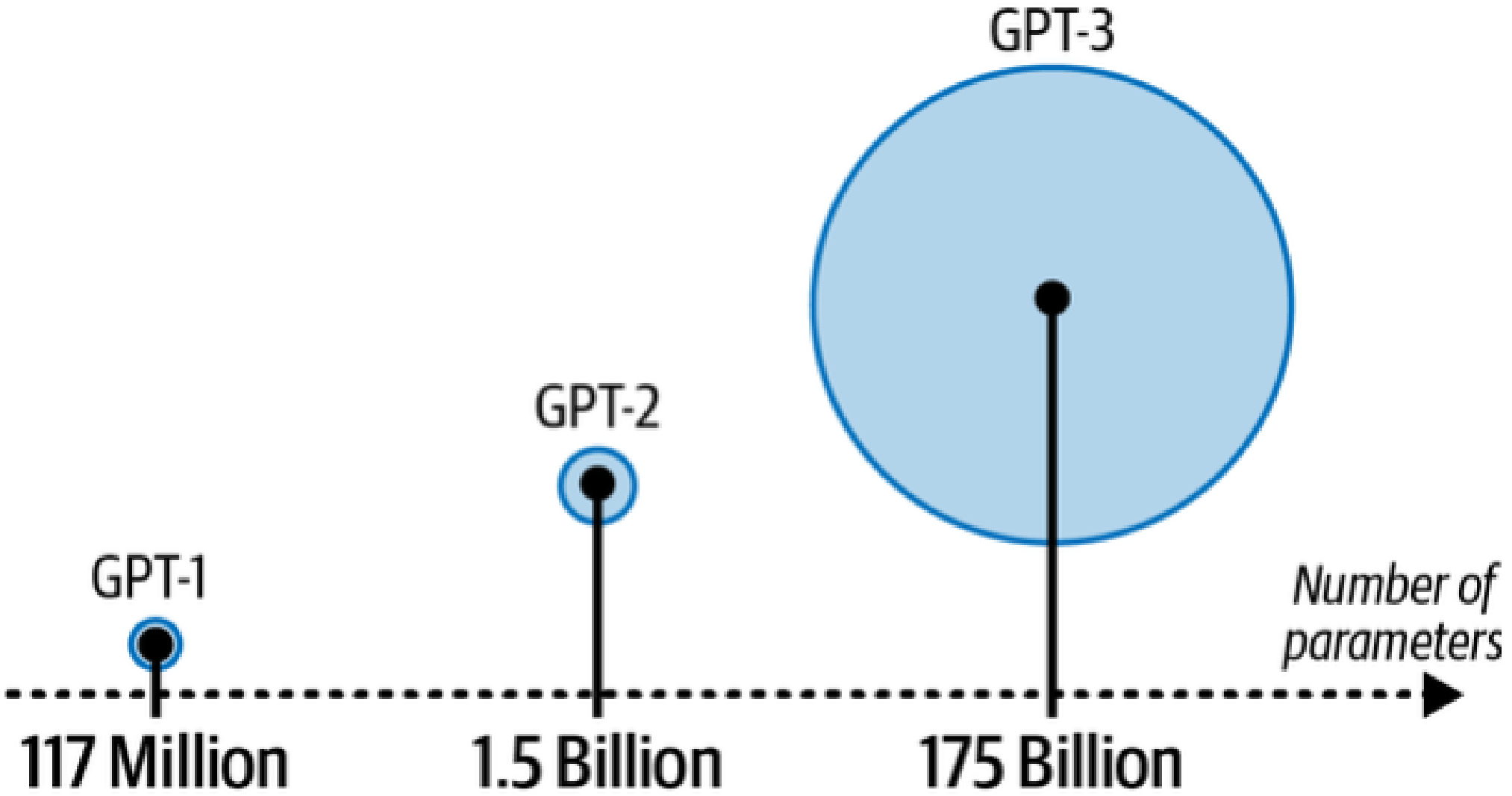


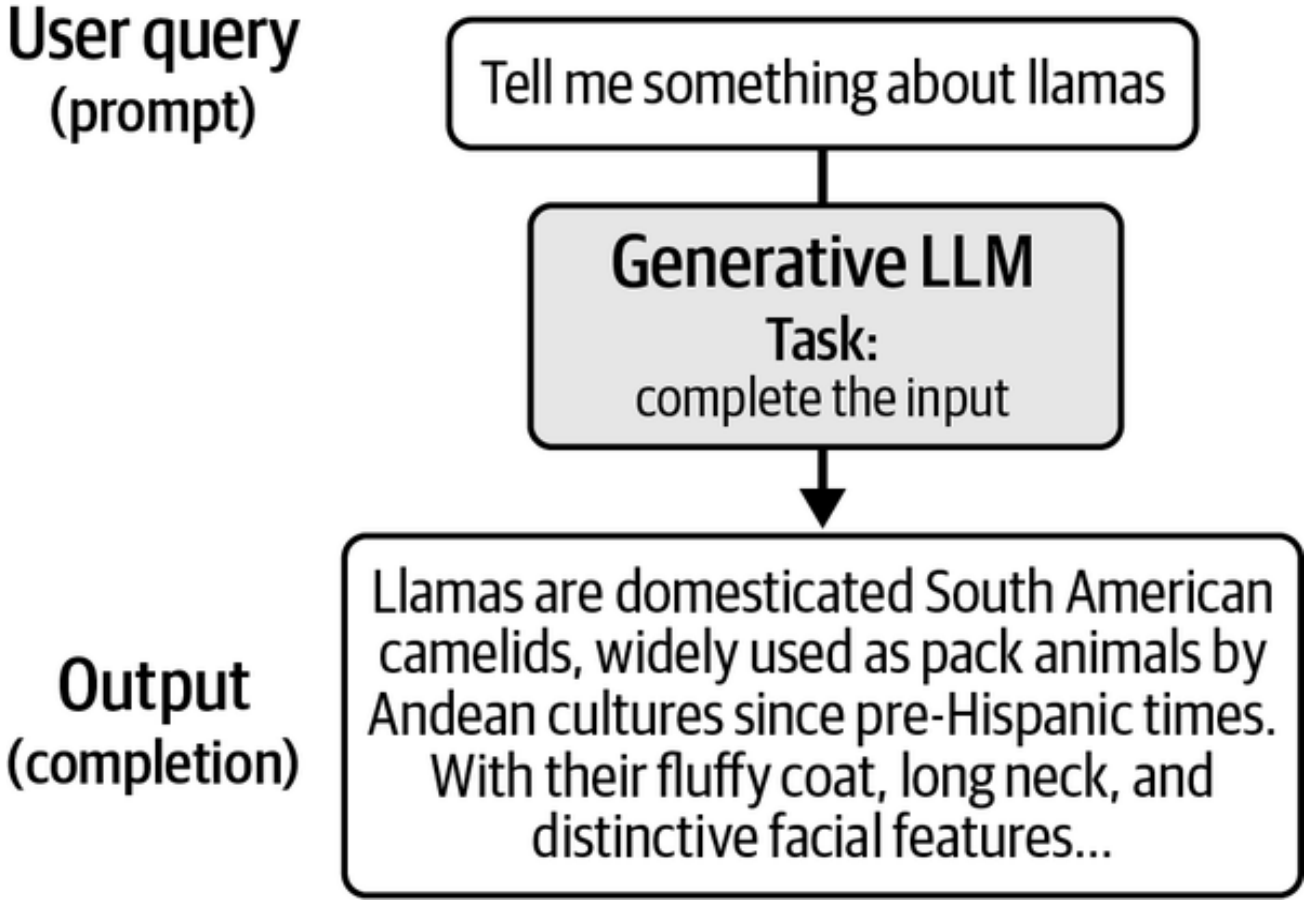
Decoder Only Models

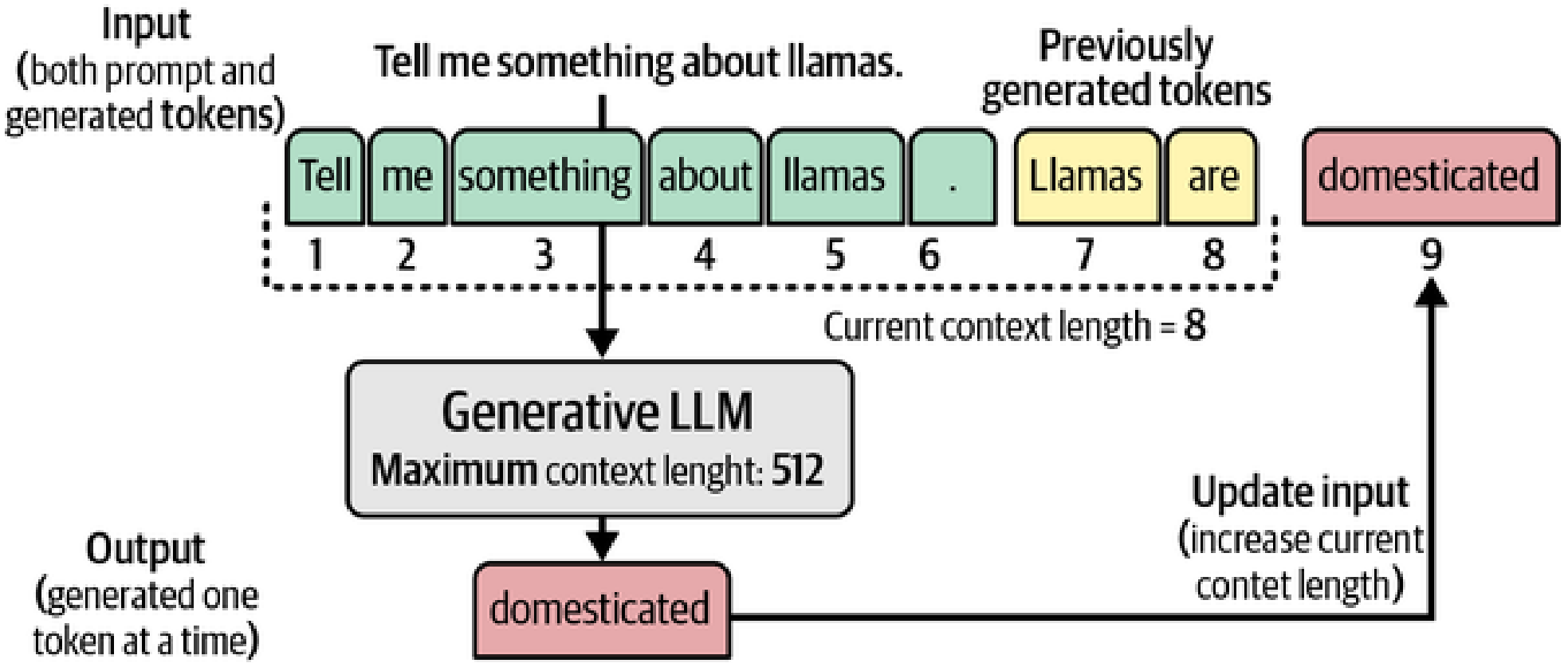
- **Purpose:**
  - Used for sequence generation or prediction tasks.
  - Examples: Text generation, language modeling, autocomplete.
- **Architecture:**
  - Multiple layers of decoders.
  - Generates output tokens step-by-step, attending to previous tokens.
  - Predicts the next token based on the context of generated tokens.

- GPT, Llama, Mistral, Claude, etc.


I
love
Input sequence
Transformer decoder
1
Transformer decoder
2
GPT-1
Transformer decoder
Masked self-attention
12
Feed-forward neural network
Next generated word
llamas


GPT-3
GPT-2
GPT-1
Number of parameters
117 Million
1.5 Billion
175 Billion


User query
(prompt)
Tell me something about llamas
Generative LLM
Task:
complete the input
Output
(completion)
Llamas are domesticated South American camelids, widely used as pack animals by Andean cultures since pre-Hispanic times. With their fluffy coat, long neck, and distinctive facial features...



Context length is the maximum number of tokens we can give the model at one time

How does the model choose the word?

Large Language Models

- Most common method for decoding in LLMs: Sampling
  - Sampling from a model's distribution over words means to choose random words according to their probability assigned by the model.
  - Iteratively choose a word to generate according to its probability in context as defined by the model.
  - More likely: generate words that the model predicts have a high probability in the context and less likely to generate words that the model predicts have a low probability.

Large Language Models

- Random sampling
- Problem:
  - mostly generating sensible, high-probable words
  - low-probability words in the tail of the distribution get chosen often enough to result in generating weird sentences.
- Instead of random sampling:
  - Use alternative sampling methods that avoid generating the very unlikely words.

Large Language Models

- Sampling methods: quality and diversity

- Most probable words tend to produce generations that are
  - more accurate
  - more coherent
  - more factual
  - but also more boring and more repetitive

Large Language Models

- Methods that give more weight to the middle-probability words:
  - are more creative
  - more diverse
  - less factual
  - more likely to be incoherent or otherwise low-quality

Large Language Models

- Top-k sampling (greedy decoding)
  - truncate the distribution of continuation words to the top k most likely words
  - renormalize to produce a legitimate probability distribution
  - randomly sample from within these k words according to their renormalized probabilities

- When k = 1, top-k sampling is identical to random sampling

Large Language Models

- Top-k sampling (greedy decoding)

- Setting k > 1 leads to
  - selected continuation word is not necessarily the most probable one
  - still probable enough
  - choice results in generating more diverse but still high-enough-quality text

Large Language Models

- Choose in advance a number of words k
- For each word in the vocabulary V , use the language model to compute the likelihood of this word given the context p(wt |w<t)
- Sort the words by their likelihood, and throw away any word that is not one of the top k most probable words.
- Renormalize the scores of the k words to be a legitimate probability distribution.
- Randomly sample a word from within these remaining k most-probable words according to its probability.

Large Language Models

- Problems with: Top-k
  - k is fixed, but the shape of the the probability distribution over words differs in different contexts
  - k = 10, sometimes the top 10 words will be very likely and include most of the probability mass, but other times the probability distribution will be flatter and the top 10 words will only include a small part of the probability mass

Large Language Models

- Nucleus or Top-p Sampling
  - keep not the top k words, but the top p percent of the probability mass
    - goal: truncate the distribution to remove very unlikely words
    - by measuring probability rather than the number of words
    - hope: the measure will be more robust in very different contexts, dynamically increasing and decreasing the pool of word candidates

Large Language Models

- Temperature Sampling
  - don’t truncate the distribution, but instead reshape it
  - Intuition:
    - thermodynamics, where a system at a high temperature is very flexible and can explore many possible states, while a system at a lower temperature is likely to explore a subset of lower energy (better) states.
- Low-temperature sampling
  - smoothly increase the probability of the most probable words and decrease the probability of the rare words

Large Language Models

- Implementation of Temperature Sampling
  - dividing the logit by a temperature parameter τ before it is normalized by passing it through the softmax in low-temperature sampling, τ Î (0, 1]
  - Instead of computing the probability distribution over the vocabulary directly from the logit, first divide the logits by τ, computing the probability vector y as:

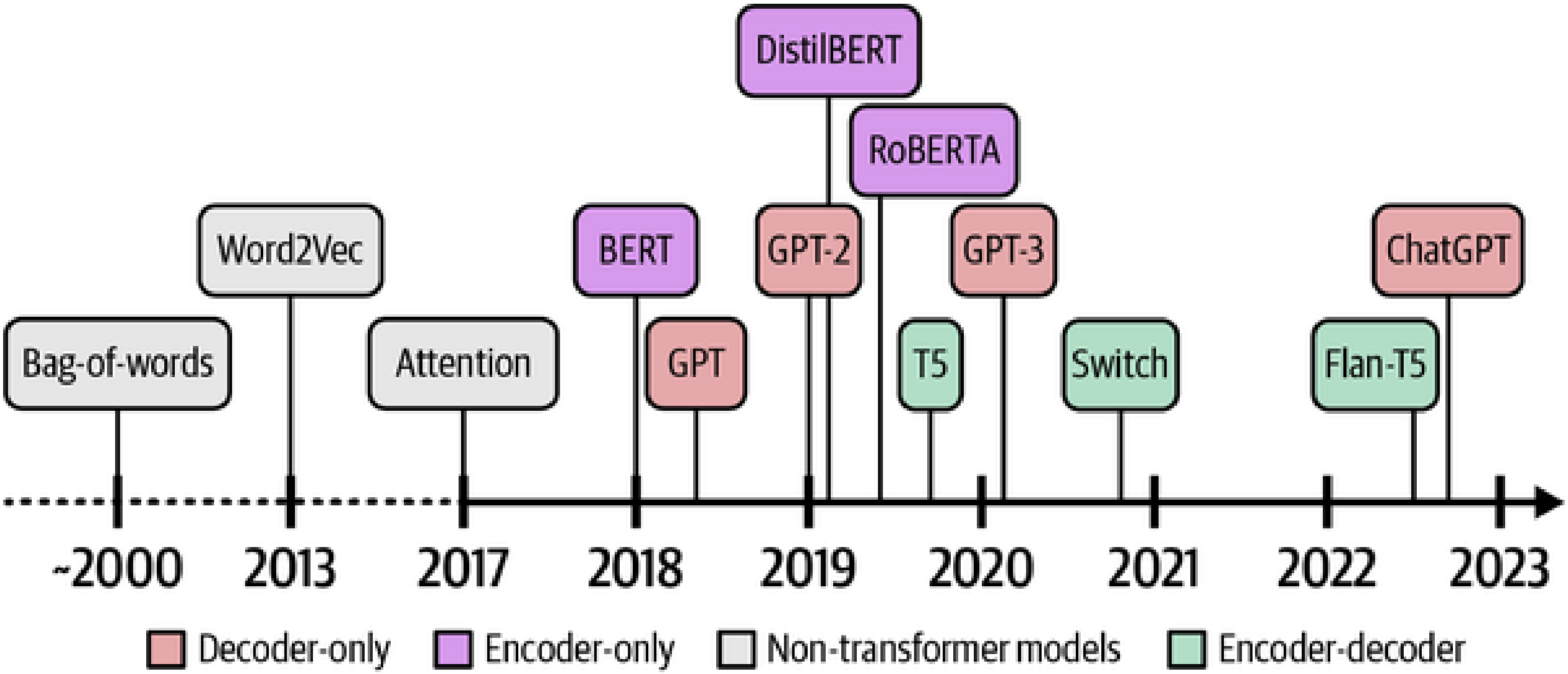
$$\mathbf{y} = \text{softmax}(u/\tau)$$

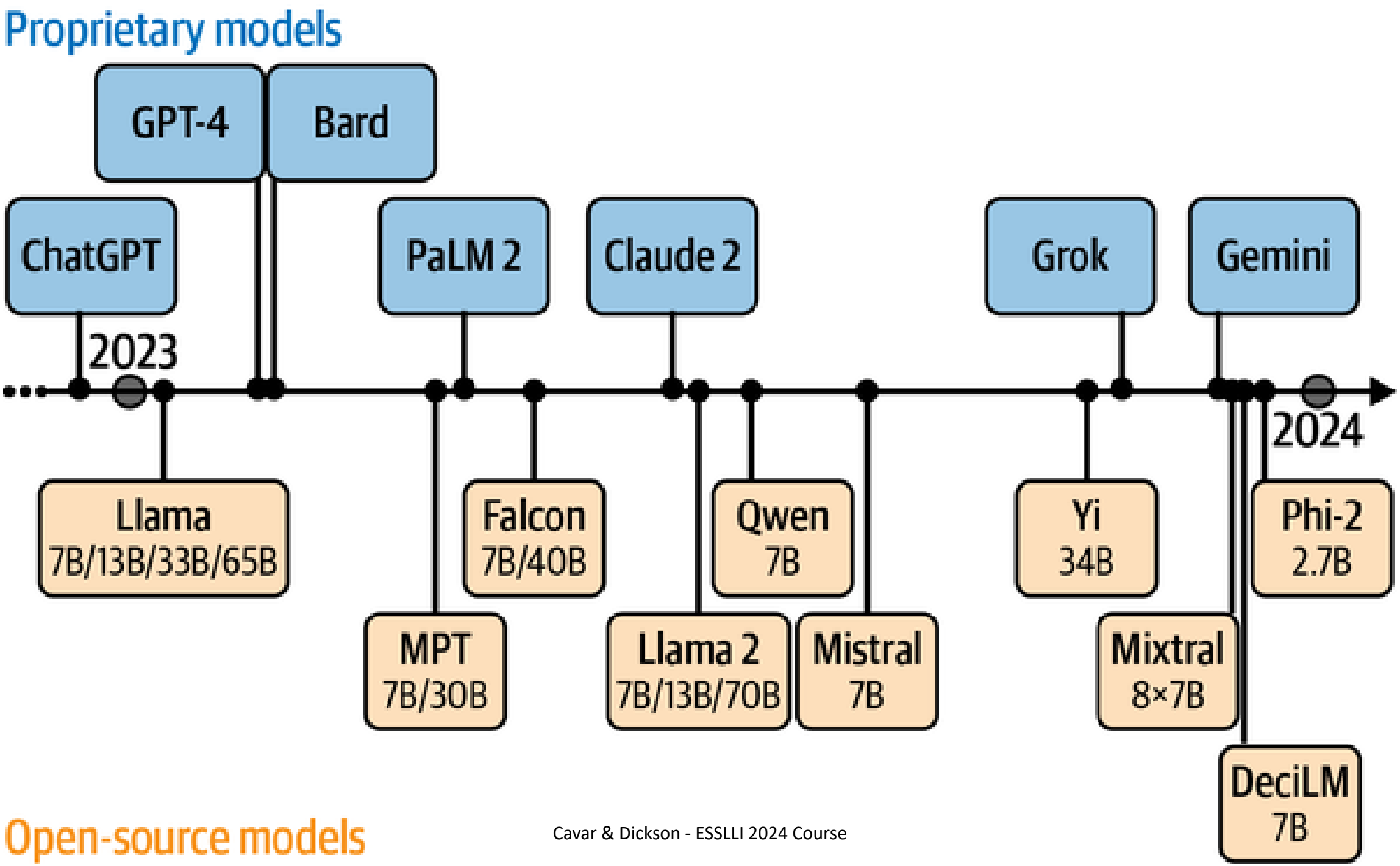
Large Language Models

- When τ is close to 1 the distribution doesn't change significantly
- The lower τ the larger the scores being passed to softmax
  - dividing by a smaller fraction $\tau \leq 1$ results in larger scores
  - softmax pushes high values toward 1 and low values toward 0
  - Larger numbers passed to a softmax result in a distribution with increased probabilities of high-probability words and decreased probabilities of low probability words
- τ closer to 0 the probability of most likely word approaches 1

Large Language Models

- We may want to flatten the word probability distribution:
  - Temperature sampling can help
  - high-temperature sampling: use $\tau > 1$.


DistilBERT
RoBERTA
Word2Vec
BERT
GPT-2
GPT-3
ChatGPT
Bag-of-words
Attention
GPT
T5
Switch
Flan-T5
~2000
2013
2017
2018
2019
2020
2021
2022
2023
Decoder-only
Encoder-only
Non-transformer models
Encoder-decoder


Proprietary models
GPT-4
Bard
ChatGPT
PaLM 2
Claude 2
Grok
Gemini
2023
2024
Llama
7B/13B/33B/65B
Falcon
7B/40B
Qwen
7B
Yi
34B
Phi-2
2.7B
MPT
7B/30B
Llama 2
7B/13B/70B
Mistral
7B
Mixtral
8×7B
DeciLM
7B
Open-source models

Meta Llama 3.1 — 8b, 70b, 405b Base & Instruct — fp16, fp8 — unsloth

15.6T tokens
128K context
Tool Multilingual

Benchmark		Llama 3 8B	Gemma 2 9B	Mistral 7B	Llama 3 70B	Mixtral 8x22B	GPT 3.5 Turbo	Llama 3 405B	Nemotron 4 340B	GPT-4 (0125)	GPT-4o	Claude 3.5 Sonnet
MMLU	(5-shot)	69.4	72.3	61.1	83.6	76.9	70.7	87.3	82.6	85.1	89.1	89.9
MMLU	(0-shot, CoT)	73.0	72.3	60.5	86.0	79.9	69.8	88.6	78.7	85.4	88.7	88.3
MMLU-Pro	(5-shot, CoT)	48.3	–	36.9	66.4	56.3	49.2	73.3	62.7	64.8	74.0	77.0
IFEval		80.4	73.6	57.6	87.5	72.7	69.9	88.6	85.1	84.3	85.6	88.0
HumanEval	(0-shot)	72.6	54.3	40.2	80.5	75.6	68.0	89.0	73.2	86.6	90.2	92.0
MBPP EvalPlus	(0-shot)	72.8	71.7	49.5	86.0	78.6	82.0	88.6	72.8	83.6	87.8	90.5
GSM8K	(8-shot, CoT)	84.5	76.7	53.2	95.1	88.2	81.6	96.8	92.3	94.2	96.1	96.4
MATH	(0-shot, CoT)	51.9	44.3	13.0	68.0	54.1	43.1	73.8	41.1	64.5	76.6	71.1

Dedicated pad_token

```
128004": {
  "content": "<|finetune_right_pad_id|>",
  "lstrip": false,
  "normalized": false,
  "rstrip": false,
```

Interesting tokens!

```
"128010": { "content": "<|python_tag|>
"128008": { "content": "<|eom_id|>"
```

Llama arch

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	6,144	12,288	20,480
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		

Llama3 RoPE Scaling

```
"rope_scaling": {
  "factor": 8.0,
  "low_freq_factor": 1.0,
  "high_freq_factor": 4.0,
  "original_max_position_embeddings": 8192,
  "rope_type": "llama3"
```

```
def apply_scaling(freqs: torch.Tensor):
    # Values obtained from grid search
    scale_factor = 8
    low_freq_factor = 1
    high_freq_factor = 4
    old_context_len = 8192  # original llama3 length
    low_freq_wavelen = old_context_len / low_freq_factor
    high_freq_wavelen = old_context_len / high_freq_factor
    new_freqs = []
    for freq in freqs:
        wavelen = 2 * math.pi / freq
        if wavelen < high_freq_wavelen:
            new_freqs.append(freq)
        elif wavelen > low_freq_wavelen:
            new_freqs.append(freq / scale_factor)
        else:
            smooth = (old_context_len / wavelen - low_freq_factor) / (
                high_freq_factor - low_freq_factor)
            new_freqs.append((1 - smooth) * freq / scale_factor + smooth * freq)
    return torch.tensor(new_freqs, dtype=freqs.dtype, device=freqs.device)
```

$$\alpha = \frac{\frac{8192}{2\pi/f} - L}{H - L} \qquad (1-\alpha)\frac{f}{s} + \alpha f$$

Chat Template

```
{% set loop_messages = messages %}
{% for message in loop_messages %}
  {% set content = '<|start_header_id|>' + message['role'] + '<|end_header_id|>\n\n' + message['content'] | trim + '<|eot_id|>' %}
  {% if loop.index0 == 0 %}
    {% set content = bos_token + content %}{% endif %}{{ content }}
{% endfor %}
{{ '<|start_header_id|>assistant<|end_header_id|>\n\n' }}
```

Data Mixture
50% General knowledge
25% Math & Reasoning
17% Code Tasks
8% Multilingual

Preprocessing
De-duplication
Heuristic filtering
fasttext, Roberta,
DistilRoberta quality

Pipeline Parallelism — 38 to 43% bfloat16 MFU

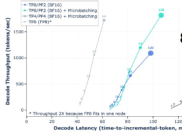
6 staged long context expansion

This long-context pre-training stage was performed using approximately 800B training tokens.

4.1.5 Model Averaging — **Languages supported:** German, French, Italian, Portuguese, Hindi, Spanish, and Thai,

Finally, we average models obtained from experiments using various versions of data or hyperparameters at each RM, SFT, or DPO stage (Izmailov et al., 2019; Wortsman et al., 2022; Li et al., 2022).

Float8 quant



7 Vision Experiments
8 Speech Experiments

	Llama 3-V 405B
MMMU (val, CoT)	64.5
VQAv2 (test-dev)	80.2
AI2 Diagram (test)	94.1
ChartQA (test, CoT)	85.8
TextVQA (val)	84.8
DocVQA (test)	92.6

@danielhanchen x.com

7/30/2024

GemmaV2: 9+27+2.6b — Base and Instruct released!

Benchmark	Mistral 7B	LLaMA-3 8B	2.6B	Gemma-2 9B	Gemma-2 27B
MMLU	62.5	66.6	51.3	71.3	75.2
MMLU Instruct		68.4		72.3	76.2

Parameters	2.6B	9B	27B
d_model	2304	3584	4608
Layers	26	42	46
pre-norm	yes	yes	yes
post-norm	yes	yes	yes
Non-linearity	GeGLU	GeGLU	GeGLU
Feedforward dim	18432	28672	73728
Head type	GQA	GQA	GQA
Num heads	8	16	32
Num KV heads	4	8	16
Head size	256	256	128
global att. span	8192	8192	8192
sliding window	4096	4096	4096
Vocab size	256128		
Tied embedding	yes	yes	yes

(1) Pre, Post MLP RMS Layernorm

NEW!
```
h = self.pre_feedforward_layernorm(h)
h = self.mlp(h)
h = self.post_feedforward_layernorm(h)
```

(5) Approx GeGLU

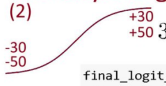
f(x), d/dx

```
x1 = self.gating_ffw(normalized_x)
x2 = self.gating_ffw_2(normalized_x)
x = gelu(x1, approximate=True) * x2
x = self.ffw_linear(x)
```

(3) Alternating SWA & Global Attn

```
sliding_window = use_swa and (i % 2 == 0)
```

(2) Grok style Logit Softcap



$$30\tanh\left(\frac{x}{30}\right)$$

```
final_logit_softcapping 30
attn_logit_softcapping  50
```

Attention Softcap 30

$$\sigma\left(30\tanh\left(\frac{1}{30}\frac{QK^T}{\sqrt{s}}\right)+M\right)V$$

(4) Different Attention Scaling

	9b	27b
head_dim =	256	128
query_pre_attn_scalar =	224	144

```
scaling = 1/sqrt(query_pre_attn_scalar)
Q *= ops.cast(scaling, dtype = Q.dtype)
```

Old $1/\sqrt{d}$
New $1/\sqrt{s}$

Logits Softcap 50

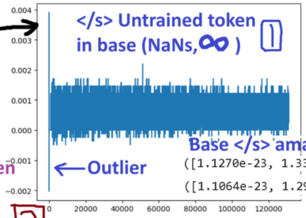
$$50\tanh\left(\frac{1}{50}XW\right)$$

Mistral Nemo 12b — 2x faster 70% less VRAM — unsloth

Mistral v3 Tokenizer = LlamaTokenizerFast
Mistral Nemo Tokenizer = GPT2TokenizerFast
Uses GPT2 Tokenizer

```
{'<unk>': 0,
 '<s>': 1,
 '</s>': 2,
 '[INST]': 3,
 '[/INST]': 4,
 '[AVAILABLE_TOOLS]': 5,
 '[/AVAILABLE_TOOLS]': 6,
 '[TOOL_RESULTS]': 7,
 '[/TOOL_RESULTS]': 8,
 '[TOOL_CALLS]': 9,
 '<pad>': 10,
 '[PREFIX]': 11,
 '[MIDDLE]': 12,
 '[SUFFIX]': 13,
 '<SPECIAL_14>': 14,
 ...
 '<SPECIAL_999>': 999}
```

Tool Usage
Pad token
Fill in the Middle
Padded Tokens



	MT Bench
Mistral 7B	6.48
Llama 3 8B	6.85
Mistral NeMo	7.84

Base </s> amax diff to instruct
([1.1270e-23, 1.3379e-01, 1.1219e-23,
([1.1064e-23, 1.2988e-01, 2.0294e-03,

(2) Auto appended EOS token (wrong)

```
tokenizer_old("Hello!").input_ids [1, 22177, 1033, 2]
tokenizer_new("Hello!").input_ids [1, 22177, 1033]
```

Mistral v3 7b

```
"head_dim": 128,
"hidden_act": "silu",
"hidden_size": 4096,
"initializer_range": 0.02,
"intermediate_size": 14336,
"max_position_embeddings": 32768,
"model_type": "mistral",
"num_attention_heads": 32,
"num_hidden_layers": 32,
"num_key_value_heads": 8,
```

Larger hidden dim

(3) New HF
```
(q_proj) (in_features=5120, out_features=4096
(k_proj) (in_features=5120, out_features=1024
```
Old HF
```
(q_proj) (in_features=5120, out_features=5120
(k_proj) (in_features=5120, out_features=1280
```

Mistral Nemo 12b

```
"head_dim": 128,
"hidden_size": 5120,
"max_position_embeddings": 1024000,
"model_type": "mistral",
"num_attention_heads": 32,
"num_hidden_layers": 40,
"num_key_value_heads": 8,
```

1M possible max position embeddings
More layers!

```
(rotary_emb.cos_cached.nbytes / 1024 / 1024)\
 * 2 * model.config.num_hidden_layers
```

Uses 20GB! Now use dynamic RoPE

Nemotron-4 340 billion

3 models: 8 trillion tokens — Base, Instruct, Reward

Number of transformer layers	Hidden dimension	Number of attention heads	Number of KV heads	Sequence length	Vocabulary size
96	18432	96	8	4096	256,000

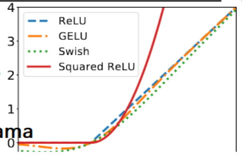
RoPE Embeddings
Sentence Piece
Squared ReLU
Untied embeddings

What is rotary %? Phi-2 style?

```
rotary_percentage: 0.5
normalization: layernorm1p
layernorm_epsilon: 1.0e-05
```

Not Llama RMS Layer Norm

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}} * \gamma + \beta$$



Squared ReLU: $\max(0, XW_{up})W_{down}$

GLU variants: $f(XW_{up}, XW_{gate})W_{down}$

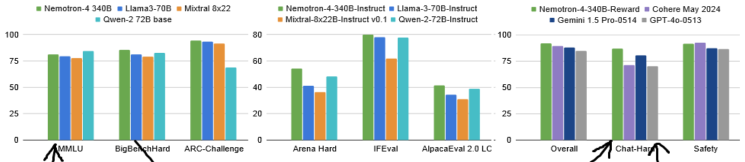
Chat template

```
<extra_id_0>System

<extra_id_1>User
{prompt 1}
<extra_id_1>Assistant
{response 1}
<extra_id_1>User
{prompt 2}
<extra_id_1>Assistant
{response 2}
```

Data-parallel size	GPUs	Iteration time (secs)	MFU (%)	Batch size	Tokens (B)
16	1536	10.3	42.4%	768	200
32	3072	10.3	42.3%	1536	200
64	6144	8.0	41.0%	2304	7600

Batch size rampup



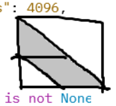
(a) Nemotron-4-340B-Base (b) Nemotron-4-340B-Instruct (c) Nemotron-4-340B-Reward

Nemotron — Nemotron — GPT-4o

Phi-3 3.8b Mini

```
"sliding_window": 2047,      "max_position_embeddings": 4096,
```

SWA like Mistral?



```
use_sliding_windows = (
    _flash_supports_window_size
    and getattr(self.config, "sliding_window", None) is not None
    and kv_seq_len > self.config.sliding_window
)
```

Upcasted RoPE like Gemma

```
q_embed = (q.float() * cos.float()) +
                      (rotate_half(q).float() * sin.float())
k_embed = (k.float() * cos.float()) +
                      (rotate_half(k).float() * sin.float())
return q_embed.to(q.dtype), k_embed.to(k.dtype)
```

Dynamic RoPE

```
class Phi3SuScaledRotaryEmbedding(_Phi3ScaledRotaryEmbedding):
math.sqrt(1 + math.log(scale) /
                math.log(self.original_max_position_embeddings))

class Phi3YarnScaledRotaryEmbedding(_Phi3ScaledRotaryEmbedding):
return 0.1 * math.log(scale) + 1.0
```

Fused MLP

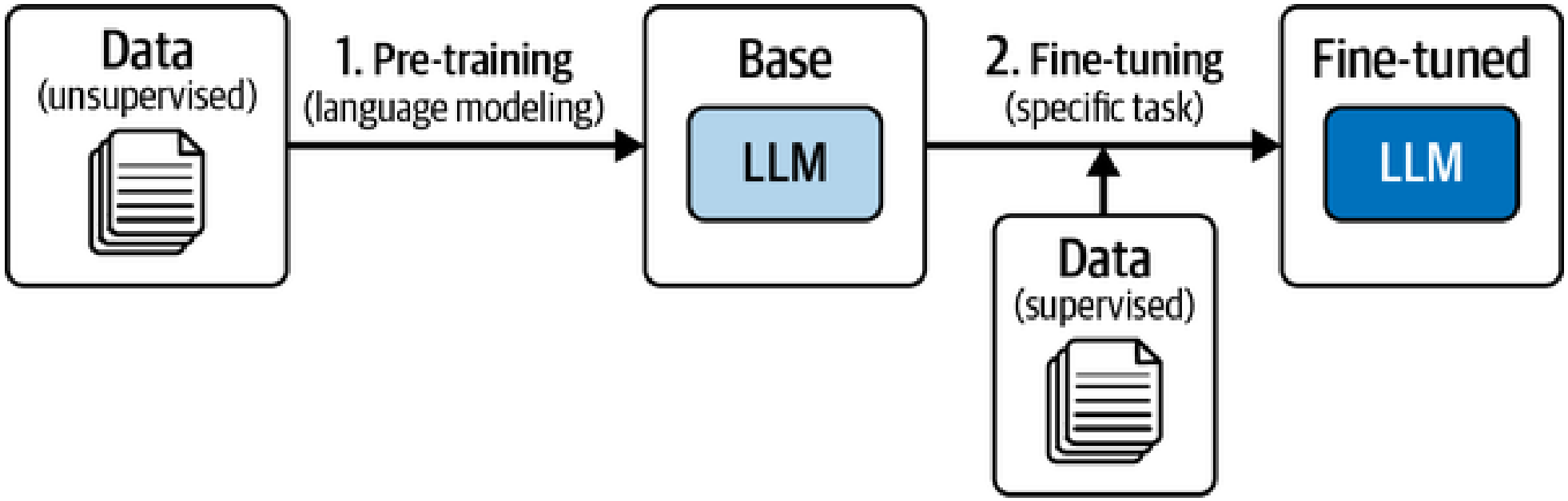
```
y = self.gate_up_proj(hidden_states)
gate, y = y.chunk(2, dim=-1)
y = y * self.activation_fn(gate)
```

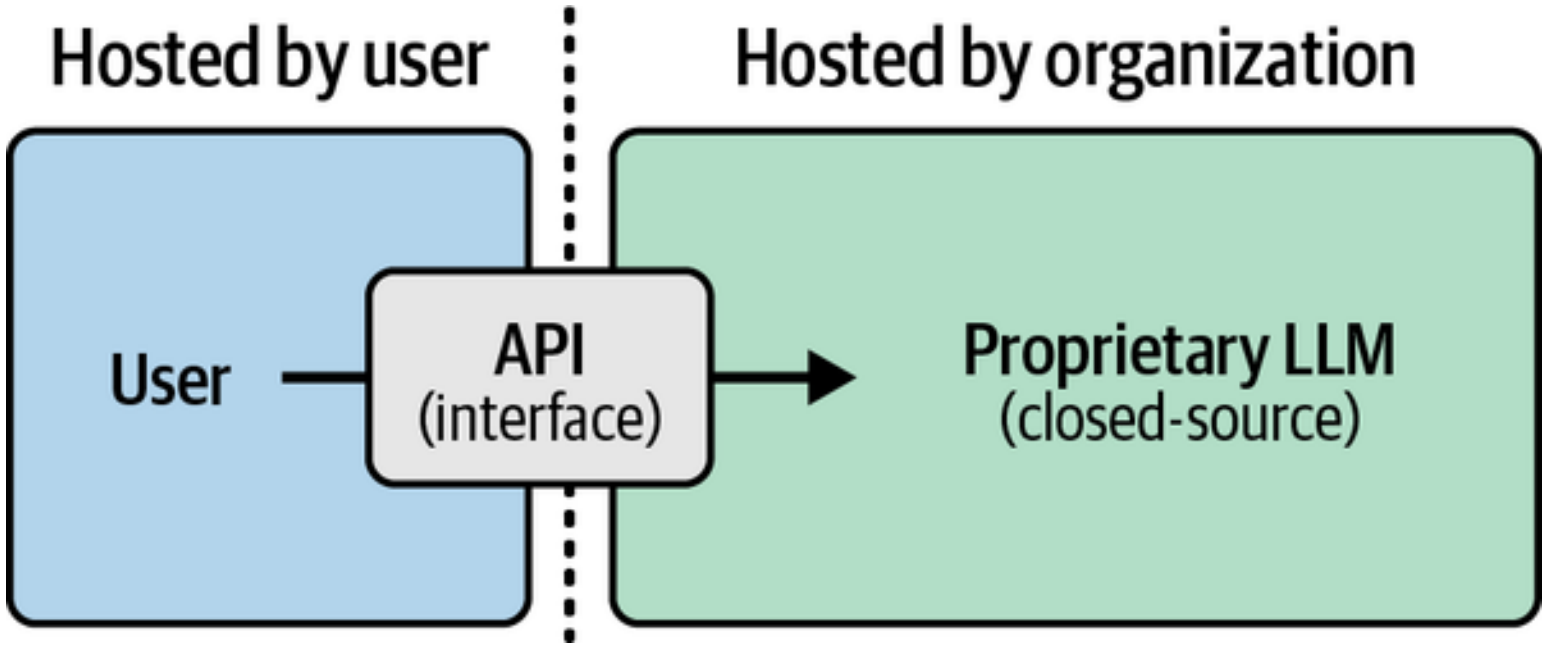
Fused QKV

```
qkv = self.qkv_proj(hidden_states)
```

Need to unfuse

	Phi-3-mini 3.8b	Mistral 7b	Gemma 7b	Llama-3-In 8b	Mixtral 8x7b	GPT-3.5 version 1106
MMLU (5-Shot) [HBK+21]	68.8	61.7	63.6	66.0	68.4	71.4
Llama-3 Evals		58.4	53.3	68.4		


Data
(unsupervised)
1. Pre-training
(language modeling)
Base
LLM
2. Fine-tuning
(specific task)
Data
(supervised)
Fine-tuned
LLM


Hosted by user
Hosted by organization
User
API
(interface)
Proprietary LLM
(closed-source)

- Reinforcement Learning with Human Feedback (RLHF)
- Positional Encoding