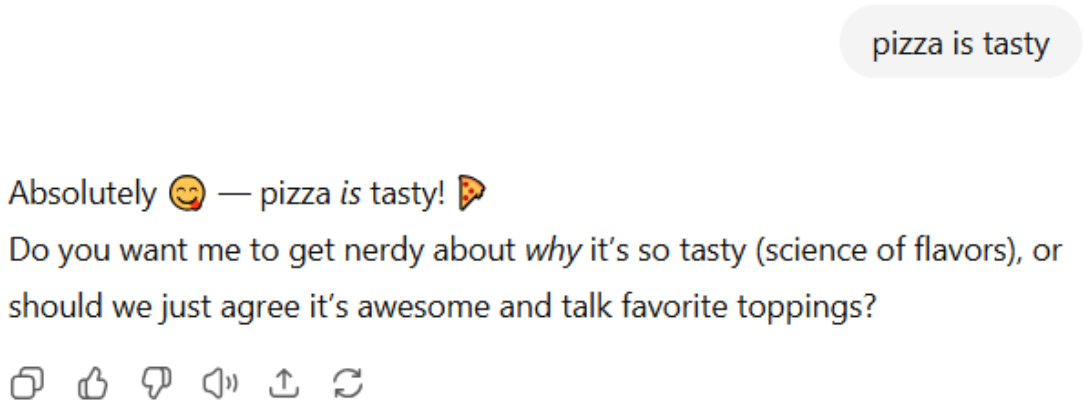
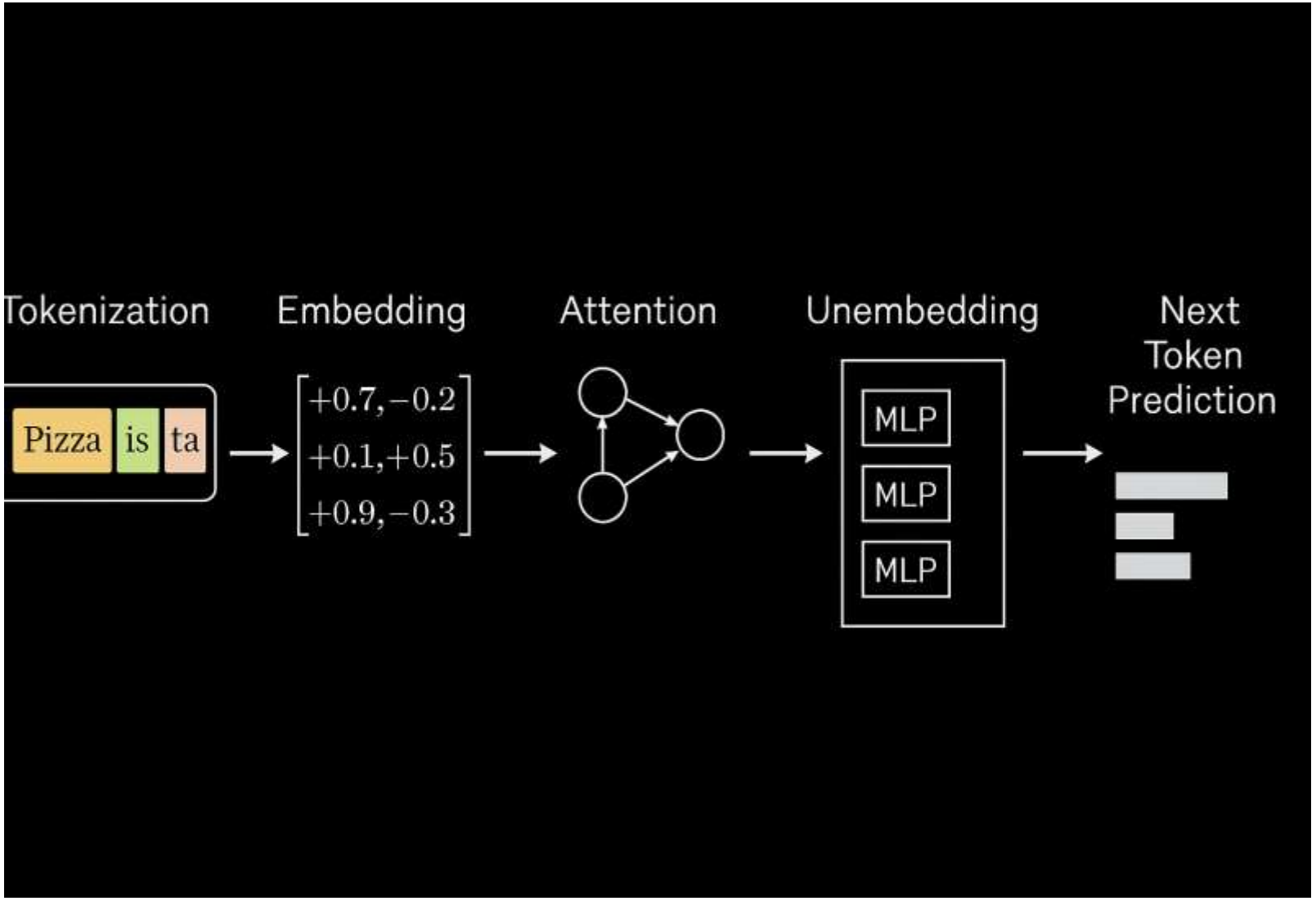
Understanding Large Language Models & Transformers

A deep dive into LLM & GPT-3

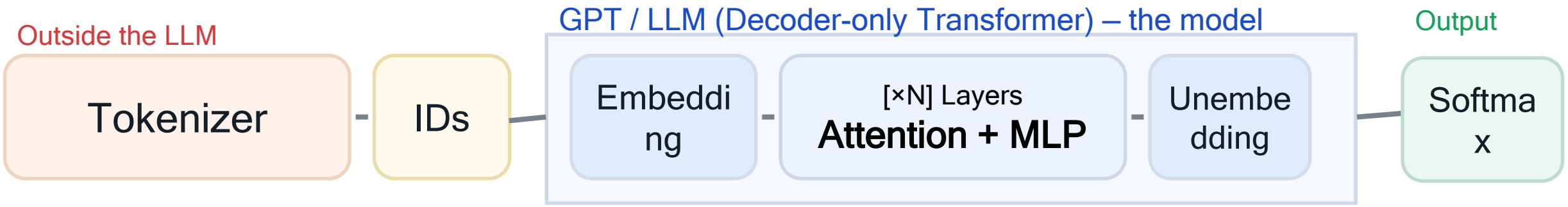
ChatGPT practical





Flow inside LLM

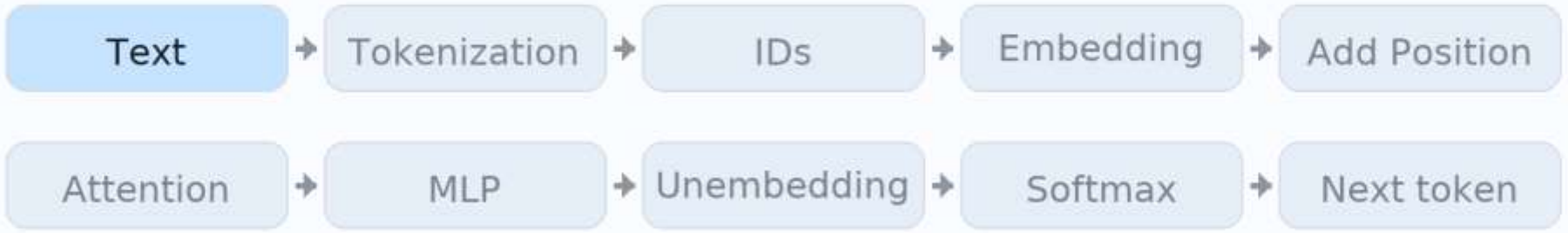
GPT is an LLM – What's the model vs. the tokenizer?



- GPT is an LLM – in this diagram, GPT = the big middle block (the model).
- Tokenizer is separate: it converts text to IDs before the model.
- Model forward pass: Embedding → [×N] (Attention + MLP) → Unembedding → Softmax.
- Pipeline: Tokenizer (outside) → GPT/LLM (the stack) → Next-token prediction (output).

Prepared by: Sai Teja Polisetty

How does text become numbers a model can understand?



Step 1: Text

Input string e.g. 'pizza'

Text: 'pizza'

Raw text

'pizza'

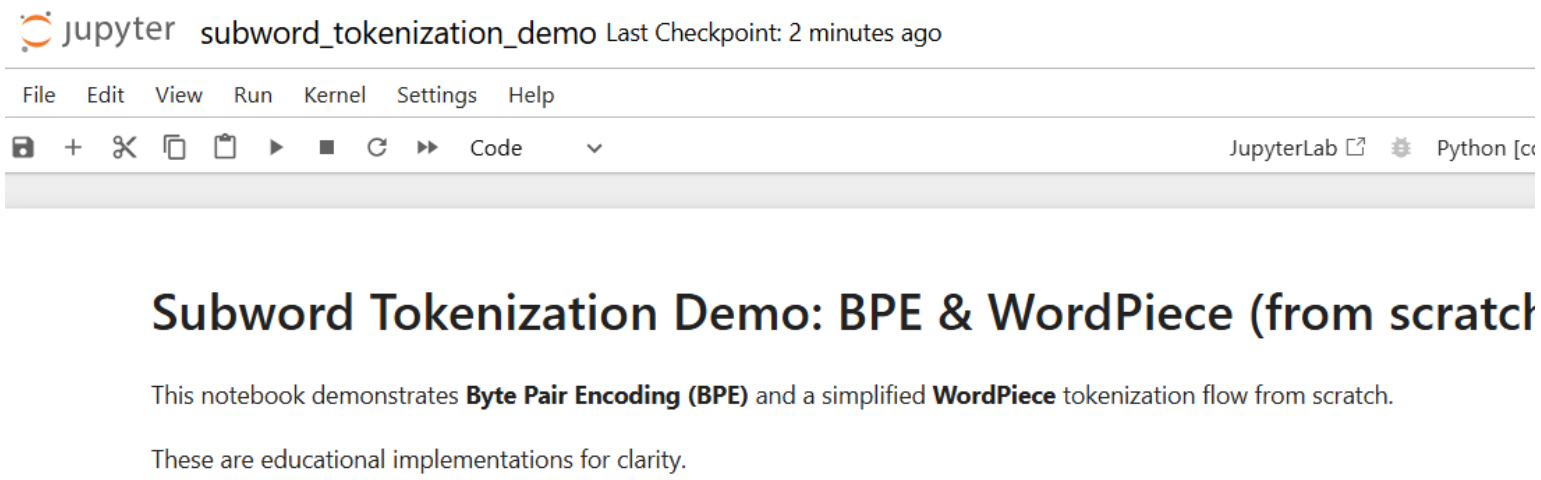
Pipeline: Text → Tokenization → IDs → Embedding + Position → Attention → MLP → Unembedding → Softmax → Next token

Prepared by: Sai Teja Polisetty

Tokenization

- Subword tokenization (BEP / WordPiece)
- Output: a list of token ids
- Example
  - “unbelievable” → ["un", "believ", "able"]
  - “I can’t” → ["I", "can", "’", "t"] or ["I", "can", "##'t"] depending on tokenizer
  - “హైదరాబాద్” → few subword chunks; highlight multilingual handling.

Python practical



Tokenization Cont...

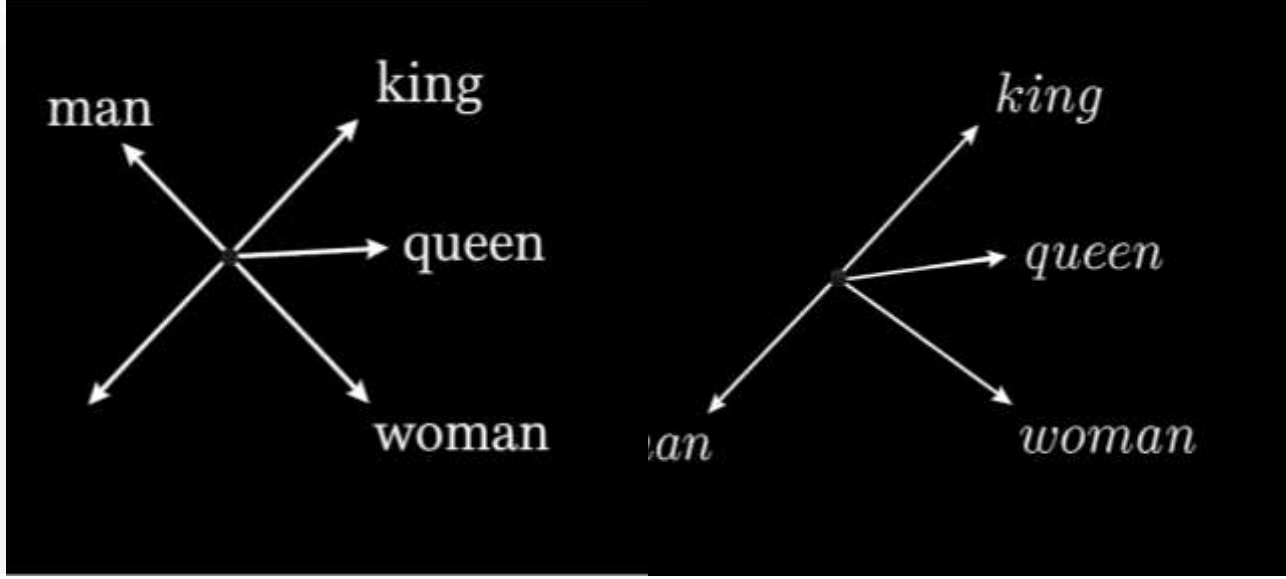
- A tokenized sequence is just an integer vector:

$$\text{ids} = [t_1, t_2, \ldots, t_n], \quad t_i \in \{0, \ldots, V-1\}$$

- where V = vocabulary size (e.g., 50,257).

Quick activity (30s)
“How does tokenization help with a misspelling like ‘beleive’?” Answer: breaks into plausible subwords—model still generalizes.

Embeddings



Dimensions = 5

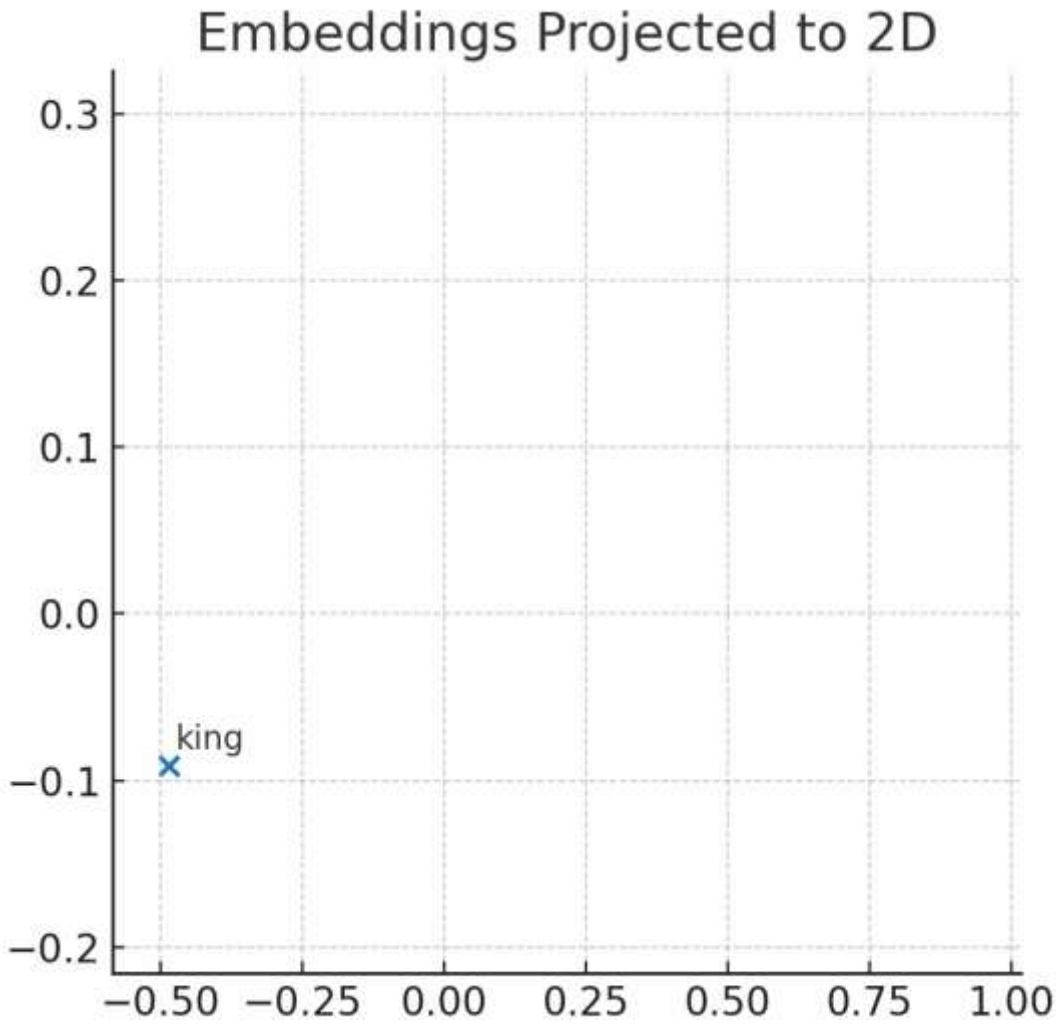
	Dim1	Dim2	Dim3	Dim4	Dim5
man	+2.31	−1.07	−0.81	−2.40	+0.23
woman	+2.49	−0.93	+1.53	+0.89	−1.12
king	+3.02	+1.48	−0.22	+0.95	+0.58
queen	+3.26	+1.84	+1.18	−0.13	−0.80

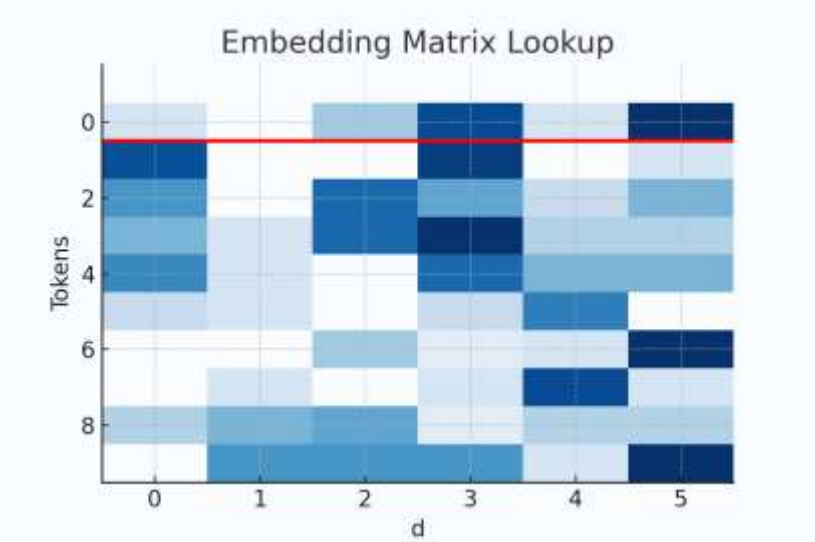
- **Embedding matrix E**: a big table of vectors–one vector per token

- **Common shapes & orientation:**
- Many frameworks store E ∈ R V×d (one row per token, d=embed dim).
- Some texts flip to d×V, Orientation doesn't change the concept; it only changes multiplication order.

- Here tokenid of woman = 2
- Vector of woman = [+2.49, -0.93, +1.53, +0.89, -1.12]

Embedding cont.... - projected in 2D



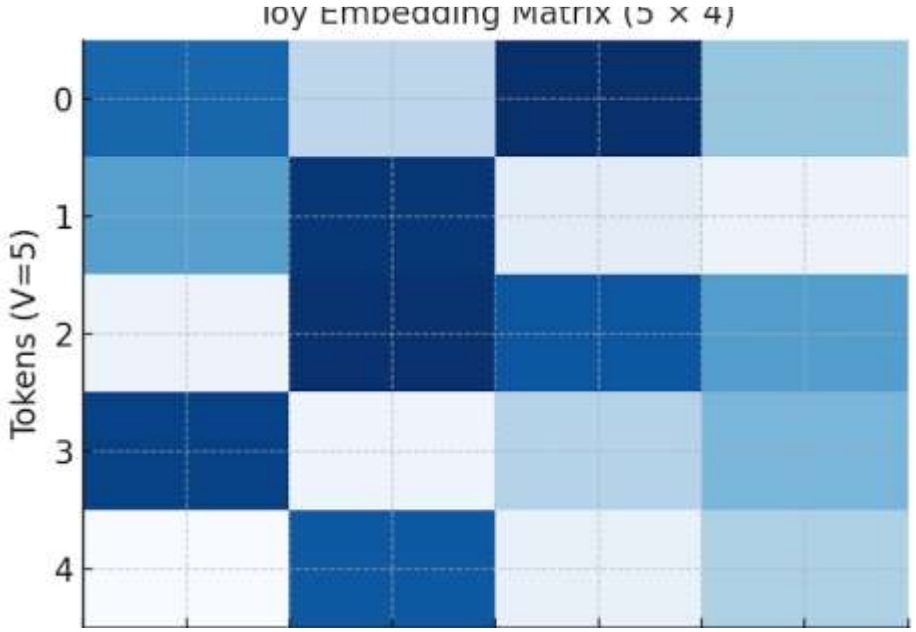


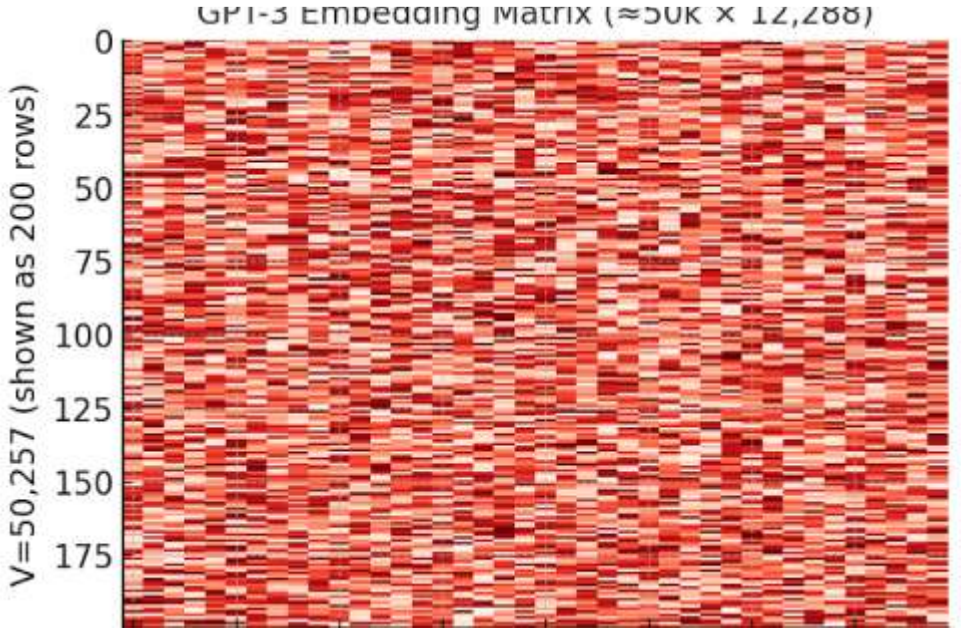
Dimensions = 5

	Dim1	Dim2	Dim3	Dim4	Dim5
man	+2.31	−1.07	−0.81	−2.40	+0.23
woman	+2.49	−0.93	+1.53	+0.89	−1.12
king	+3.02	+1.48	−0.22	+0.95	+0.58
queen	+3.26	+1.84	+1.18	−0.13	−0.80

Embedding cont....

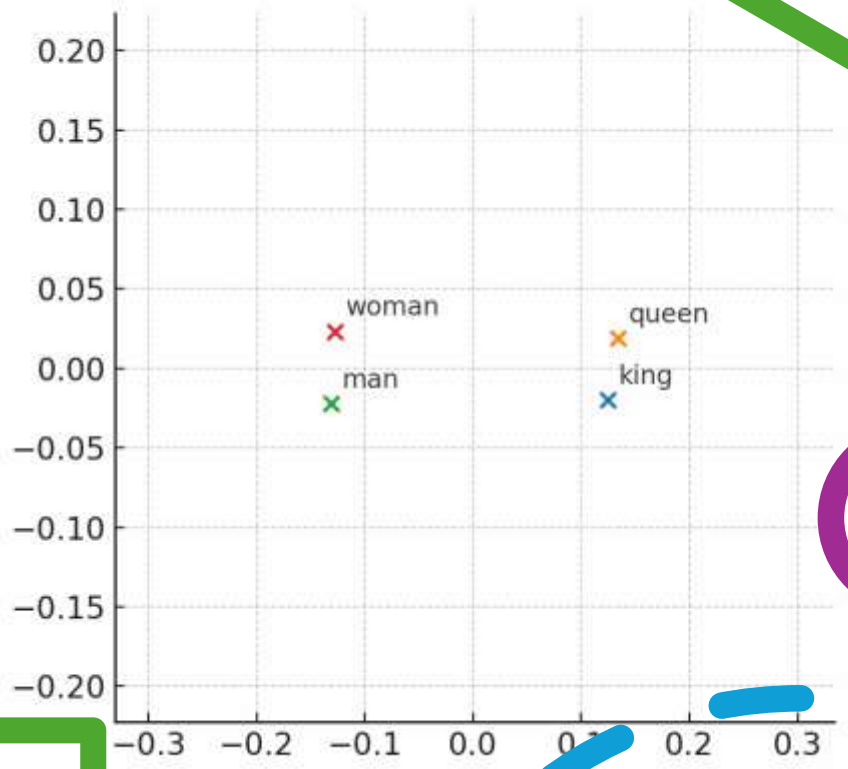
- Token ID's -> Embeddings
  - Each token id is used to look up a row in the embedding matrix.
  - Example:
    - Vocab size V=4
    - Embedding dimension d=5
    - Embedding matrix shape = V×d
  - For token id 2, we grab row 2.
    - That row is a vector with 5 numbers.





Embedding cont.... - comparison with GPT-3

- Token ID's -> Embeddings
  - Each token id is used to look up a row in the embedding matrix.
  - Example:
    - Vocab size V=50,257 | Embedding dimension d=12,288 | Embedding matrix shape = V×d
  - For token id 12345, we grab row 12345.
    - That row is a vector with 12,288 numbers.



Embedding cont... - similarity

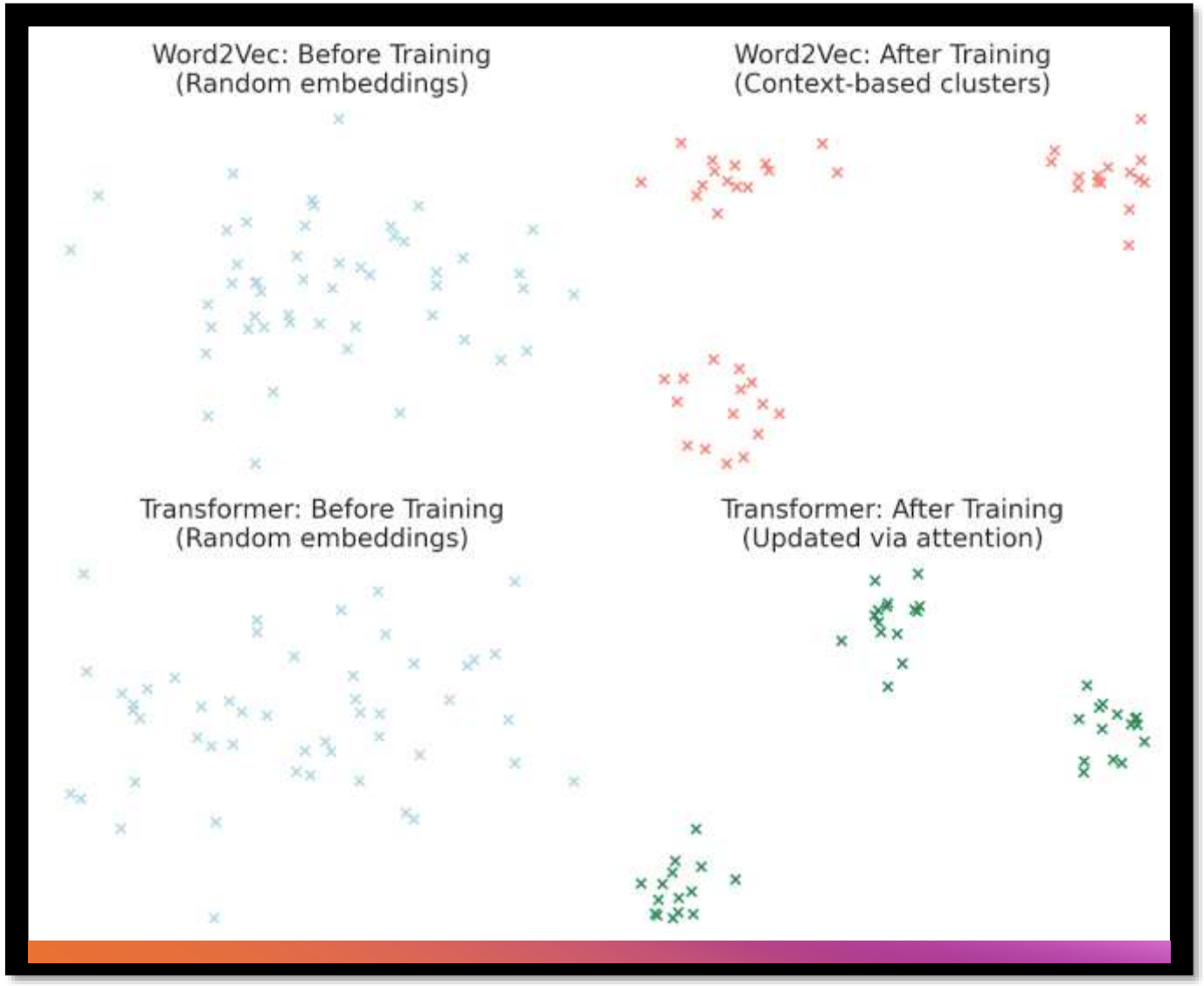
- Cosine similarity

$$\cos\_\text{sim}(a, b) = \frac{a \cdot b}{\|a\| \, \|b\|}$$

- Dot product (a · b): measures how much two vectors point in the same direction.
- Denominator : normalizes by their lengths, so we only care about direction, not magnitude.
- Range: between -1 and 1.
  - +1 = exactly same direction → very similar meaning.
  - 0 = orthogonal → unrelated.
  - -1 = opposite direction → opposite meaning.
  - “king” and “queen” vectors form a small angle → cosine ≈ 1.
  - “king” and “dog” vectors are far apart → cosine ≈ 0.
  - “hot” and “cold” might point opposite directions → cosine ≈ -1.
- Intution examples:
  - Analogies: queen≈king−man+woman
  - Clustering: vadapav sits near street-food tokens; burger near fast-food; show them in two clusters.

Embeddings cont.... - why it matters?

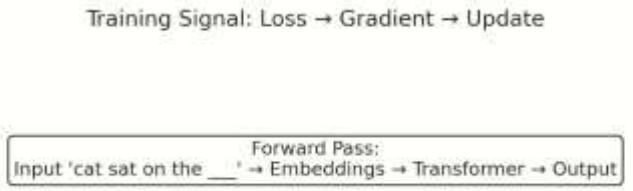
- Embeddings encode meaning prior to attention–like a semantic starting point.
- During training, embedding vectors move (via backprop) so that similar words land closer; dissimilar words move apart.
- Downstream, attention will mix these vectors contextually. But the raw meaning starts here.

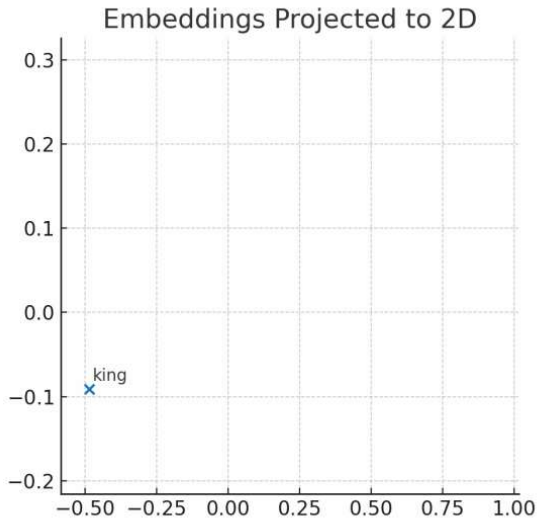


Embeddings cont.... - why it matters?

Aspect	Standalone Word Embeddings (Word2Vec, GloVe)	Embeddings in Transformers (GPT, BERT, etc.)
Training setup	Only embeddings are trained (no attention, no transformer).	Embedding matrix is the first layer of the transformer. Training is end-to-end (embeddings + attention + MLP).
How training works	Model learns embeddings by predicting context words (e.g., Word2Vec Skip-gram: “pizza” → predict “cheese,” “oven,” etc.).	Gradients from attention + feed-forward layers backpropagate into the embeddings, updating them indirectly.
Resulting embeddings	Words with similar contexts cluster together purely due to embedding training.	Embeddings improve because they co-evolve with attention + MLP layers during pretraining.
Clustering behavior	Clustering comes only from the embedding training process.	Clustering emerges as embeddings + attention + MLPs adapt jointly.
Isolation	Embeddings are trained in isolation (separate models like Word2Vec, GloVe).	Embeddings are not isolated — they are updated as part of the whole transformer.

$$E \leftarrow E - \eta \cdot \frac{\partial \mathcal{L}}{\partial E}$$





Embeddings cont.... - why it matters?

- Embedding evolution: Word2Vec → GloVe → Transformers.
- Training signal: Loss →→ gradient →→ update E.

Python practical:

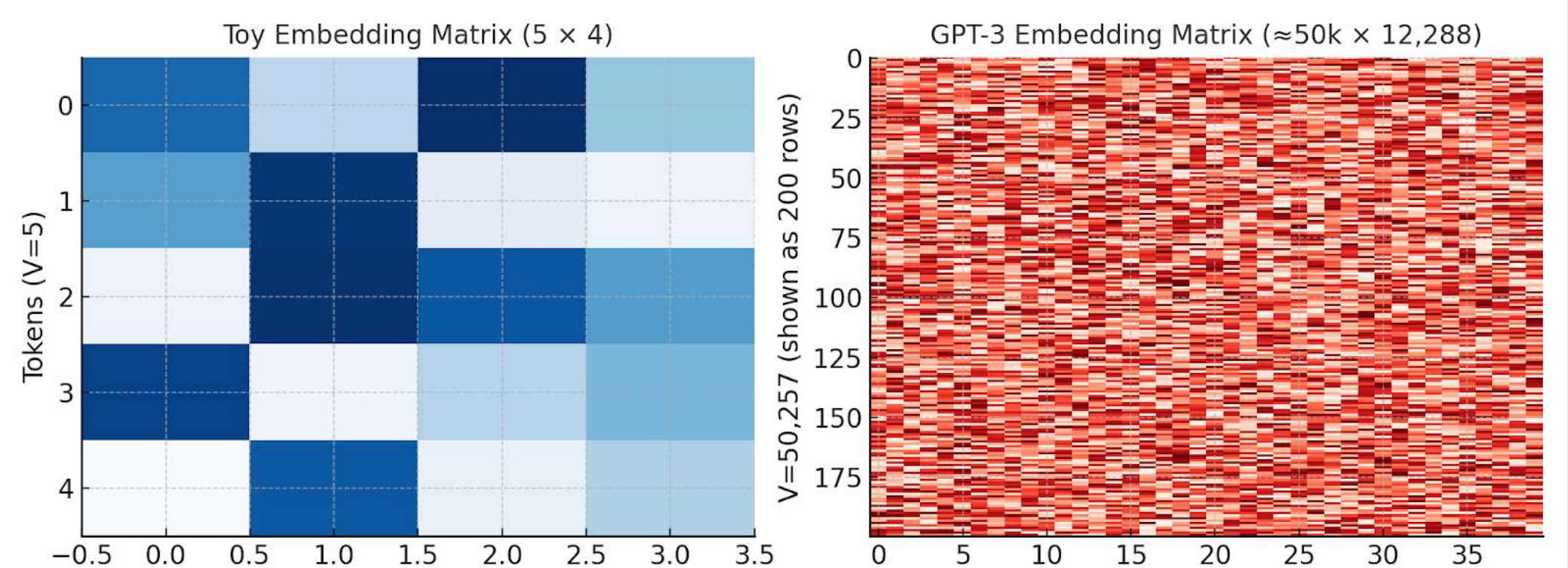
- Semantic closeness = vector closeness
  - Similar words cluster in vector space.
  - Example: queen ↔ king, queen ↔ woman.
- Analogical reasoning emerges
  - king - man + woman ≈ queen.
- Irrelevant words stay far away
  - "chef" is correctly distant from "queen".
  - This shows embeddings separate concepts (royalty vs food).

[1]:
```
# Toy embeddings (d=4) for a few tokens
E = {
  "king":  [0.9, 0.8, 0.1, 0.0],
  "queen": [0.88, 0.82, 0.12, 0.02],
  "man":   [0.75, 0.60, 0.05, -0.02],
  "woman": [0.72, 0.62, 0.07,  0.00],
  "chef":  [0.10, 0.22, 0.85,  0.40],
}
```

[3]:
```
def cos_sim(a,b):
    import math
    dot = sum(x*y for x,y in zip(a,b))
    na  = math.sqrt(sum(x*x for x in a))
    nb  = math.sqrt(sum(y*y for y in b))
    return dot/(na*nb)
```

[5]:
```
 # Nearest neighbors for 'queen'
sorted([(w, cos_sim(E["queen"], v)) for w,v in E.items() if w!="queen"],
       key=lambda x: x[1], reverse=True)
```

[5]:
```
[('king', 0.9994525047495437),
 ('woman', 0.9987652835589712),
 ('man', 0.9953584400375494),
 ('chef', 0.3226748570228522)]
```

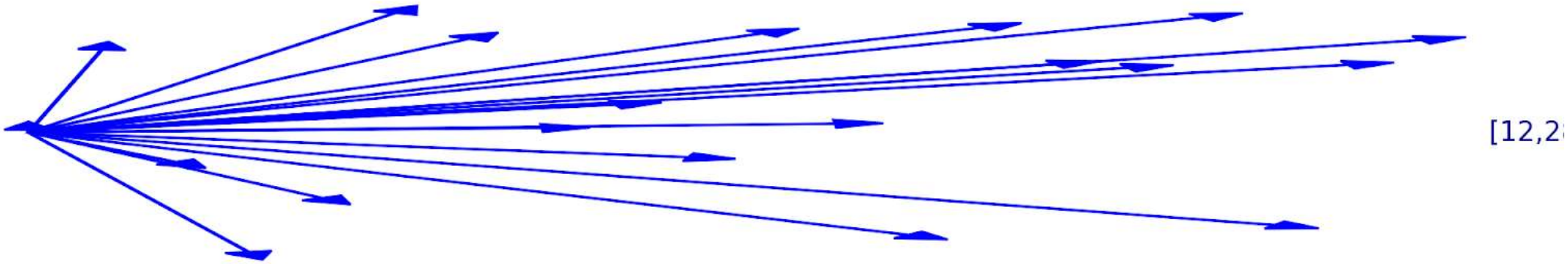


Connect to the "big model" numbers

- GPT-3’s tokenizer knows about ≈50,257 tokens. These cover words, subwords,punctuation, emojis, even parts of rare words.”

Connect to the "big model" numbers cont ….

- Each token is mapped to a 12,288-dimensional vector. That means every token lives in a huge high-dimensional space – far beyond human intuition.



So the embedding matrix has shape: V×d=50,257×12,288 ≈ 617,558,016 weights. 617 million params.

At the other end, we have the unembedding layer. It mirrors the input: shape d×V.This converts the final hidden vector back into scores for all tokens in the vocabulary.

Think of embeddings as the entry door, and unembeddings as the exit door of the model. Everything else – attention and MLP layers – happens in between.

Embedding (V × d) → [Transformer Layers] → Unembedding (d × V)

Connect to the "big model" numbers cont ….

- Total weights = 175,181,291,520
- Organized into 27,938 matrices

Matrix name	Matrix dimension
Embedding	D_embed * n_vocab 12,288 * 50,257 = 617558016
Key	
Query	
Value	
Output	
Up-projection	
Down-projection	
UnEmbedding	n_vocab * D_embed 50,257 * 12288 = 617558016

Weights covered so far:

617558016
+ 617558016
==========
1,23,51,16,032
==========

Remaining to study:

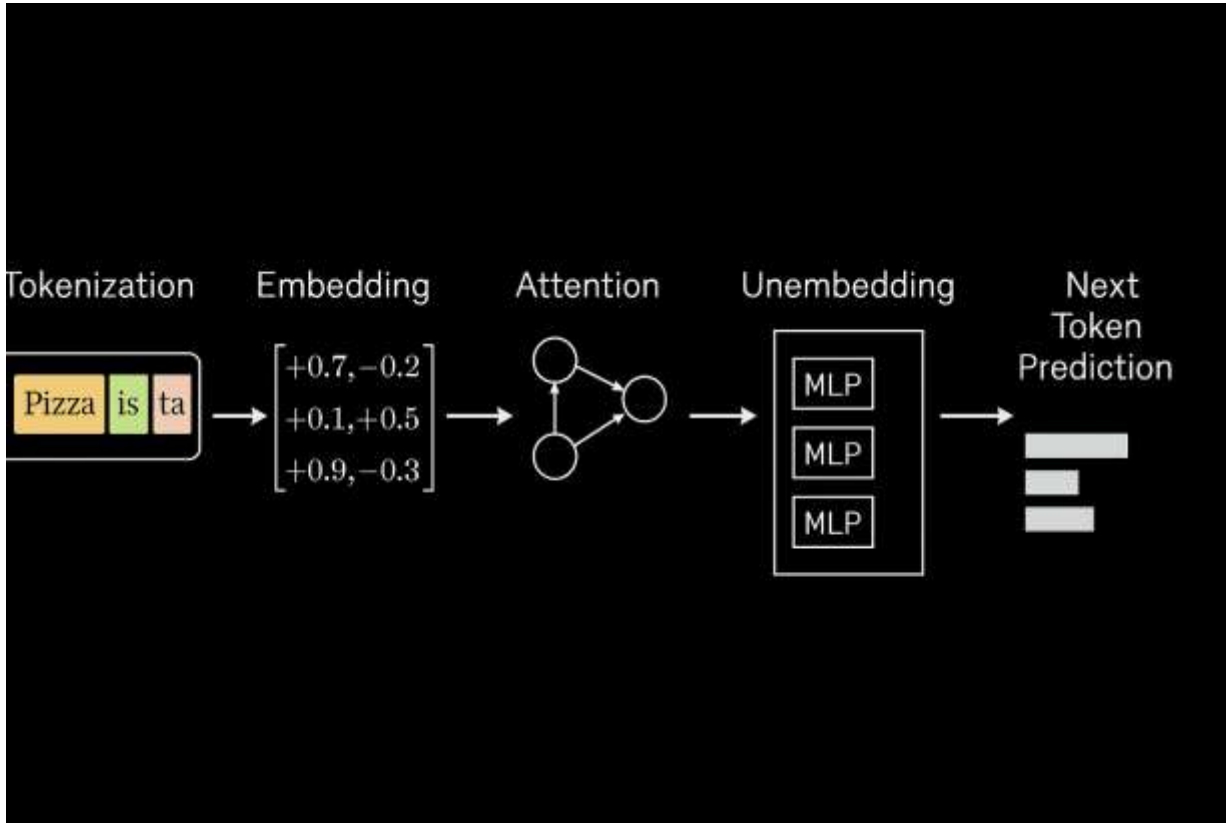
175,181,291,520
- 1,23,51,16,032
==========
173,946,175,488
==========

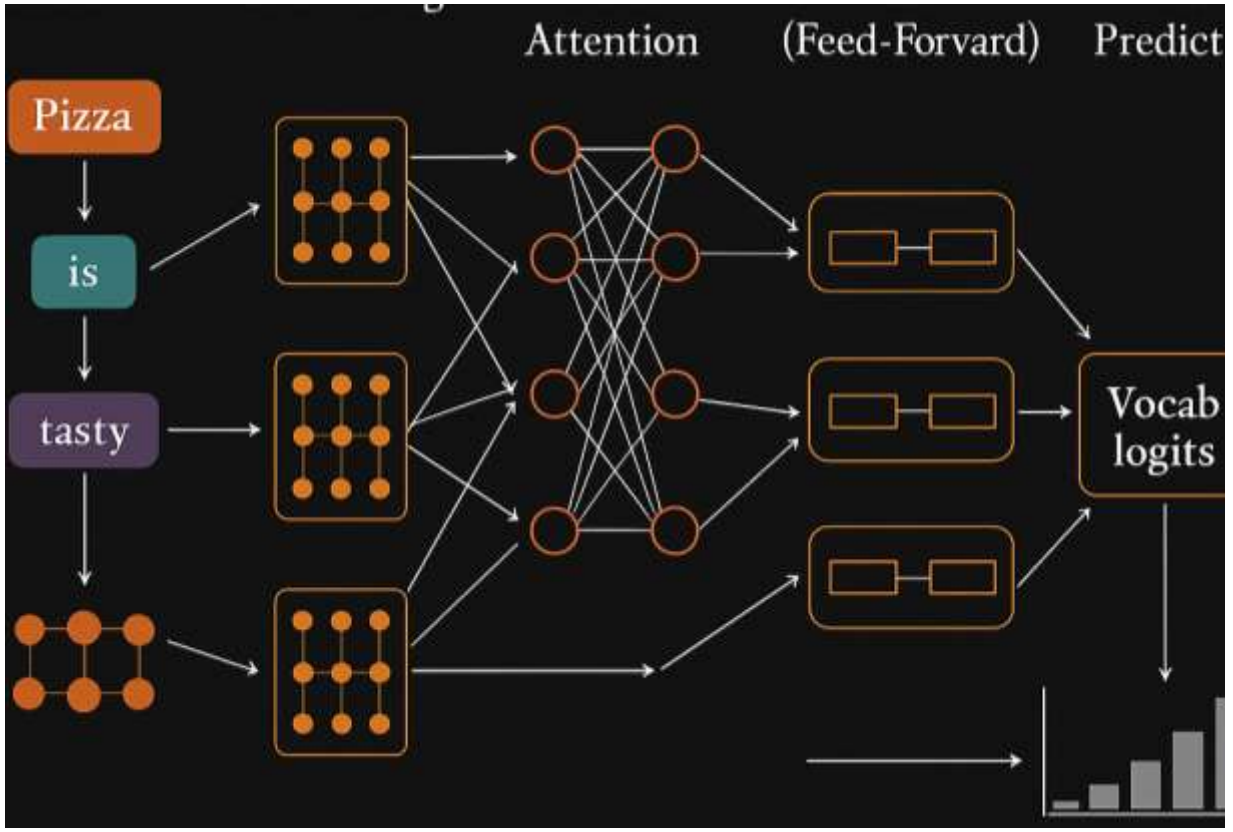
Quiz Time

Q1: Why subword tokenization instead of whole words?

Q2: What does cosine similarity capture?

Q3: Does it matter if we store tokens as rows or columns? (Answer: convention only.)


Tokenization
Embedding
Attention
Unembedding
Next Token Prediction
Pizza
is
ta
+0.7, −0.2
+0.1, +0.5
+0.9, −0.3
MLP
MLP
MLP


Attention
(Feed-Forvard)
Predict
Pizza
is
tasty
Vocab logits

Summary