Embeddings In LLM

An embedding is a numerical representation of tokens (words, sub-words or characters)

Instead of treating tokens as arbitrary IDs, embeddings place them in a high-dimensional vector space where semantic meaning is captured.

How It Works

1. Tokenization → IDs

   Text is first tokenized into tokens and mapped to integer IDs.

   Example: "Arfin" → ID = 101

2. Embedding Layer

   Each ID is mapped to a dense vector of real numbers (eg., 768-dim or 4096-dim depending on the model).

Example (simplified):

"king" → [0.21, -0.57, 0.88, ...]

"queen" → [0.19, -0.55, 0.90, ...]

Semantic Proximity

Words with similar meanings end up closer in vector space.

Famous example:

king - man + woman = queen

Usage In LLMs

Embeddings are the input layer to the Transformer

They preserve both syntactic (grammer) and semantic (meaning) properties.

Later layers (attention + feed-forward) refine these embeddings to capture context.

Types of Embeddings In LLM

Token Embeddings → Convert each token ID into a dense vector.

Positional Embeddings → Add information about token order (since Transformers don't have sequence awareness by default).

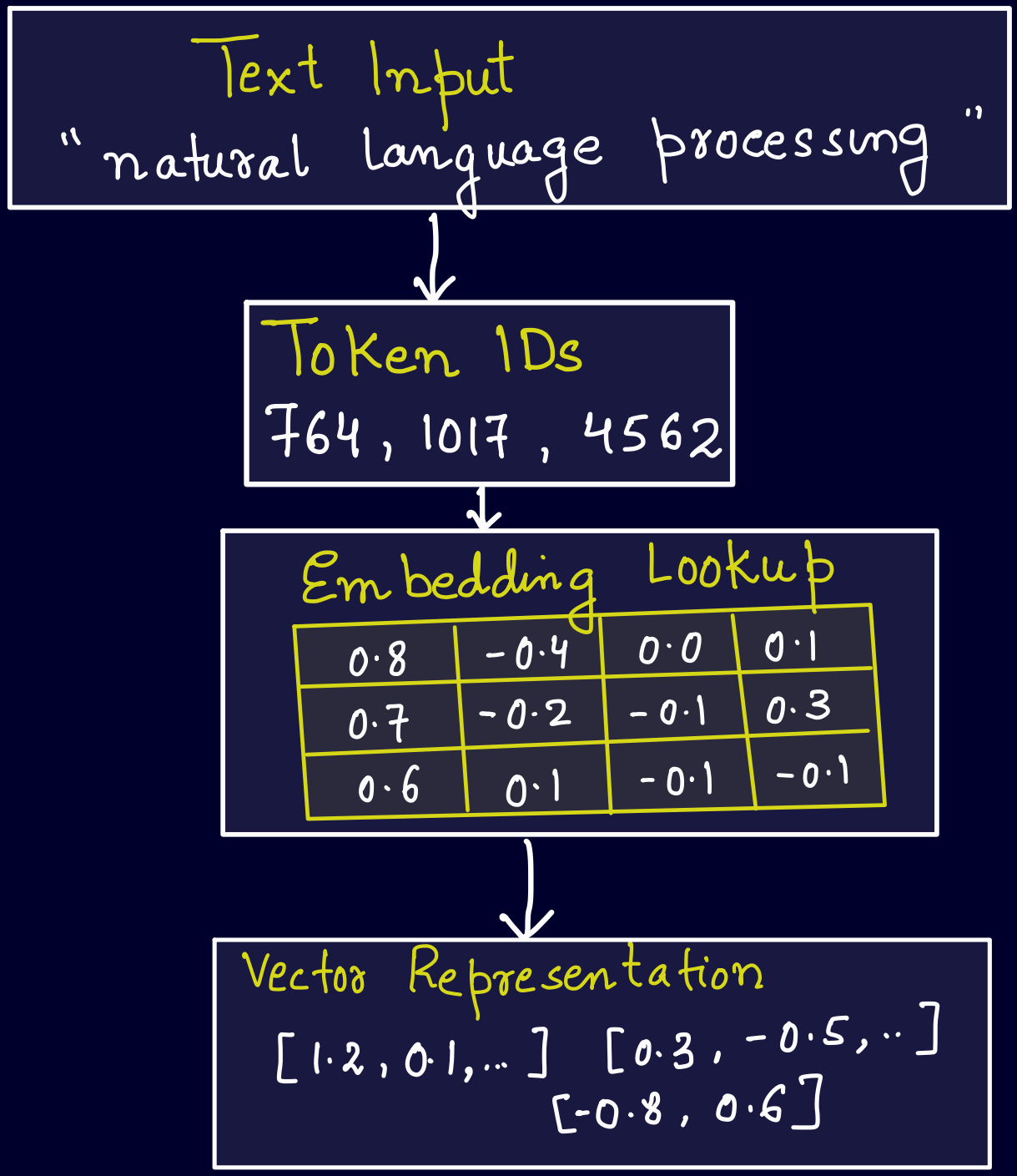
Contextual Embeddings → Evolve as text passes through layers, capturing word meaning in context.

Why Embeddings Matters

They enable LLMs to generalize beyond exact word matches.

Compactly encode relationships, analogies, and context.

Serve as the foundation for tasks like Semantic search, clustering, recommendation and retrieval - augmented generation (RAG)


Text Input
" natural language processing "
Token IDs
764 , 1017 , 4562
Embedding Lookup
0·8 -0·4 0·0 0·1
0·7 -0·2 -0·1 0·3
0·6 0·1 -0·1 -0·1
Vector Representation
[1·2 , 0·1 , ...] [0·3 , -0·5 , ..]
[-0·8 , 0·6]

An embedding is a dense vector representation of discrete tokens (words, subwords, characters).

Formally: An embedding is a mapping

$$f : V \rightarrow R^d$$

Embedding layer implemented as a look up table (a learnable weight matrix):

$$E \in R^{|V| \times d}$$