Introduction to LLM for Text Generation

Module-1

What is a Language Model?

- A machine-learning model that predicts the probability of the next token given its context
- **Scale matters:** Modern LLMs have hundreds of millions to hundreds of billions of parameters. More training data + compute = better performance on complex language tasks.
- **What Are Text Generation Models?**

Text generation is the process of creating readable and meaningful text by predicting the next word or sequence of words, using patterns learned from large amounts of text data .

Core Techniques used: Rule-based, Statistical model, Neural/Transformer-based models.

Vector Representation

- **From text to numbers:** Natural language must be converted to numbers for models to process.
- LLMs convert text into numerical vectors using a process called tokenization and embedding.
- Vectors capture semantic relationships and enable reasoning over meaning. Similar words will have vectors that are closer together in the vector space.
- LLMs can perform mathematical operations on these vectors to understand relationships between different pieces of text, enabling tasks like similarity search, text generation, and more.

Vector Representation

- **Word Embeddings (Static):** The process begins with tokenizing the input text into smaller units, such as words or subwords. Each token is then mapped to a high-dimensional space, where similar tokens are placed closer together. This mapping enables the model to understand context and semantics, supporting tasks like semantic search, text classification, and sentiment analysis. The generated embeddings are then stored in the vector database for efficient retrieval and analysis.
- **Word2Vec (2013):** learns word vectors based on context, encodes semantic similarity ("king" ≈ "queen"). Word2vec uses shallow neural networks, typically with one hidden layer, to learn these embeddings. There are two main architectures: (CBOW/Skip-Gram).

  Continuous Bag-of-Words (CBOW): Predicts a target word based on its context.

  Skip-gram: Predicts the surrounding context words given a target word.
- **GloVe (2014):** uses global co-occurrence matrices, capturing linear word relationships ("king – man + woman = queen"). GloVe relies on the frequency with which words appear together in a text corpus to create these vector representations.

Vector Representation

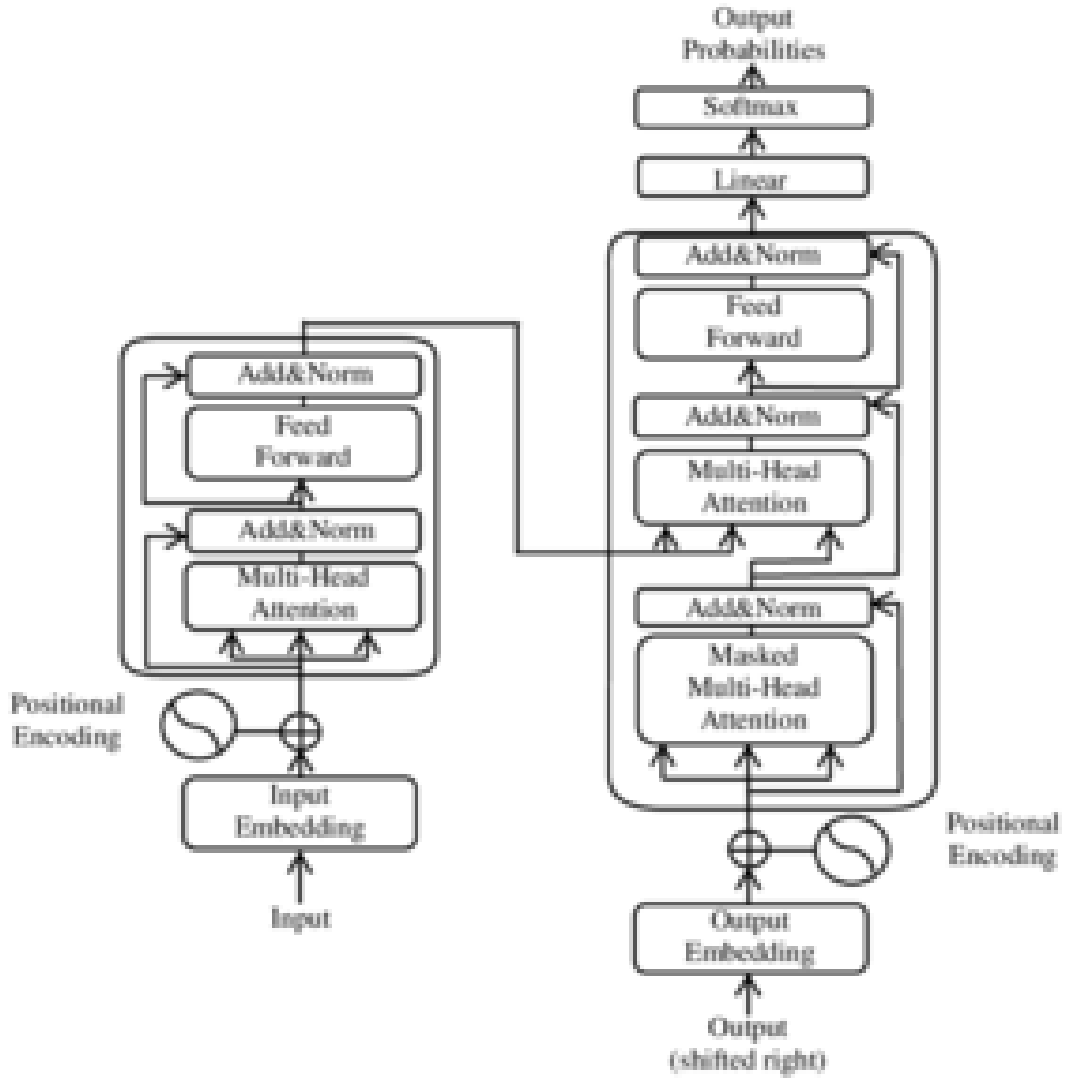
- **Contextual embeddings:** Contextual embeddings in NLP are word representations that adapt their meaning based on the surrounding context, unlike traditional embeddings that assign a single, static vector to each word. This means the same word can have different embeddings depending on the sentence it's used in, capturing nuances and relationships that static embeddings miss.
- Generated by models like BERT/GPT that factor in surrounding context
- **ELMo (2018):** bidirectional LSTM produces context-specific word vectors
- **BERT (2018):** transformer-based, masked-language training yields dynamic token embeddings per context

Vector Representation

- **How Vectors Work in Practice?**
- Words/sentences mapped to **high-dimensional numeric vectors**.
- Similar meaning → vectors close in cosine/euclidean space
- **Dimension Trade-Offs ---**
- **Higher dimensions =** more nuance, but heavier compute/memory costs
- Choose based on task complexity and resource constraints.
- **Indexing & Retrieval ---**
- **Approximate Nearest Neighbor (ANN):** HNSW graphs speed up similarity search
- Critical for scalable retrieval in RAG and chat memory systems.

Transformer Architecture

- “Attention Is All You Need” (2017) Architecture
- BERT Encoder
- GPT decoder



Transformer Architecture

- **Core innovations:**
- **Self-attention (Scaled dot-product):** $\text{softmax}(QK^T/\sqrt{d_k})V$
- **Multi-head attention:** captures diverse contextual relationships
- **Positional encoding:** uses sin/cos waves to encode sequence order
- **Encoder vs Decoder:**
- **Encoder-only** (e.g., BERT): great for understanding/contextual tasks
- **Decoder-only** (e.g., GPT family): used for generating text autoregressively
- **Encoder–Decoder** (e.g., T5): versatile – excellent for translation, summarization

Probabilistic Text Generation

- LLMs generate text **token by token**, treating each next token as a random variable drawn from a probability distribution predicted by the model based on prior context
- This approach introduces **stochasticity and variation**, allowing richer, more human-like outputs.
- The model produces ***logits*** – raw scores – for every token in its vocabulary.
- These logits are converted into a probability distribution using **softmax**, making higher scores more likely selections
- Decoding Strategies: uses Greedy Search, Beam Search

Probabilistic Text Generation

- **Sampling Methods ----**
- **Random Sampling:**
- Picks any token based on its probability, allowing full randomness.
- Highly variable but often incoherent
- **Temperature Sampling**
- Modifies distribution sharpness:
  - High T (>1): flattens probabilities → more exploration
  - Low T (<1): sharpens → more conservative output
- **Top-k Sampling**
- Restricts options to the top k tokens, then samples from that set.
- Smaller k → more focused; larger k → more creativity
- **Top-p (Nucleus) Sampling**
- Dynamically selects the smallest set whose cumulative probability ≥ p (e.g. 0.9), then samples from that pool
- Adapts to the distribution shape – more flexible and coherent outcomes.

Probabilistic Text Generation

making their *interpretations more consistent*.

Probabilistic Text Generation: The Decision Mechanism

After the transformer understands the context of the given text, it moves on to generating new text, guided by the concept of likelihood or probability. In mathematical terms, the model calculates how likely each possible next word is to follow the current sequence of words and picks the one that is most likely:

$$w_{next} = \text{argmax}\, P(w \mid w_1, w_2, ..., w_m)$$

By repeating this process, as shown in Figure 2-3, the model generates a coherent and contextually relevant string of text as its output.

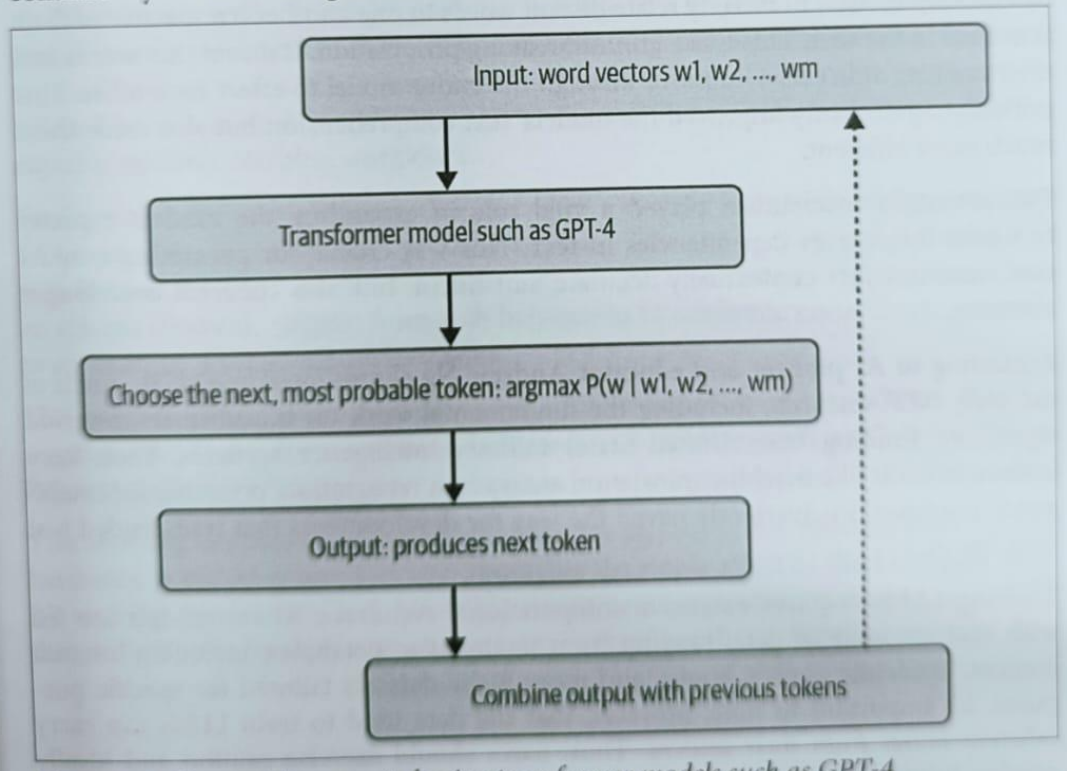


Figure 2-3. How text is generated using transformer models such as GPT-4