READ MODE · ANSWER FIRST

HOW LLMs WORK

An interactive, visual explanation of HOW LLMs WORK, from setup through the complete system.

CHEAT SHEET · 6 ESSENTIAL IDEAS

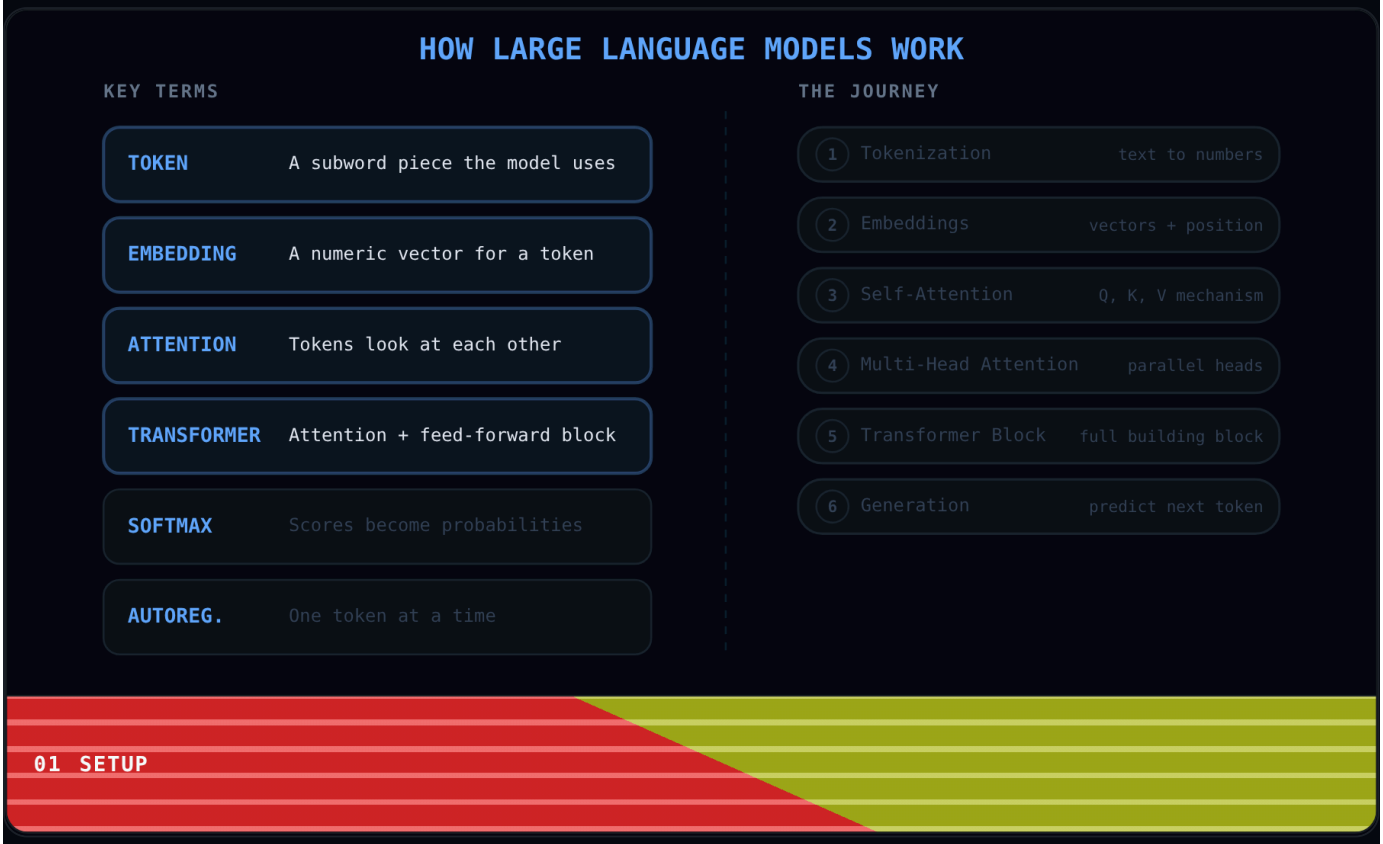
The whole story in 6 lines

HOW LLMs WORK becomes easier to reason about when every stage is connected as one system.

1. Pipeline grammar: text string → BPE splits → token ID lookup. Progressive pill reveal.
2. Two-tier: embedding table lookup on top, positional encoding addition below. Structural scene.
3. Illustrative scene: Q/K/V projection, attention score matrix with softmax, weighted value sum. This is the most detailed technical stage.
4. Fanout grammar: input fans out to N parallel heads, results concatenate and project. Each head shows a mini attention pattern.
5. Flow scene: vertical pipeline with residual skip connections shown as bypass arrows. Add&Norm layers between major components.
6. Layered scene: stack of N transformer blocks, then linear projection + softmax over vocabulary + sampling. Temperature control.

STAGE 1 OF 8

Setup



Welcome. This explainer walks you through the internals of a large language model, from raw text all the way to a generated response. No prior machine learning knowledge is required.

A Token is a small chunk of text. Models do not read characters or whole words. They split text into subword pieces called tokens. The word "unbelievable" might become three tokens: "un", "believ", "able".

An Embedding is a list of numbers (a vector) that represents a token. These vectors capture meaning. Tokens with similar meanings end up with similar vectors.

Attention is the mechanism that lets each token look at every other token in the sequence. It is how the model learns that "it" in "The cat sat because it was tired" refers to "cat".

A Transformer is the building block of modern LLMs. It combines attention with a feed-forward network and uses skip connections to let information flow easily through deep stacks.

Softmax is a function that turns a list of raw scores into probabilities that sum to one. It appears twice in the architecture: inside attention, and at the very end when the model picks the next token.

Now that you know the vocabulary, here is the journey ahead. We will follow a piece of text through six stages: tokenization, embedding, self-attention, multi-head attention, the full transformer block, and finally token generation. Let us begin with how raw text becomes numbers the model can process.

STAGE 2 OF 8

Tokenization

02 TOKENIZATION

Here is the question this stage answers: how does a model turn human text into numbers it can actually compute with? The answer is tokenization.

The input text arrives as a raw string of characters. The model cannot work with characters directly. It needs a fixed vocabulary of subword pieces that balance between whole words and single characters.

The tokenizer scans the text and splits it using merge rules learned during training. Byte-Pair Encoding (BPE) starts with individual characters and repeatedly merges the most frequent pairs. Common words stay whole. Rare words get split into familiar pieces.

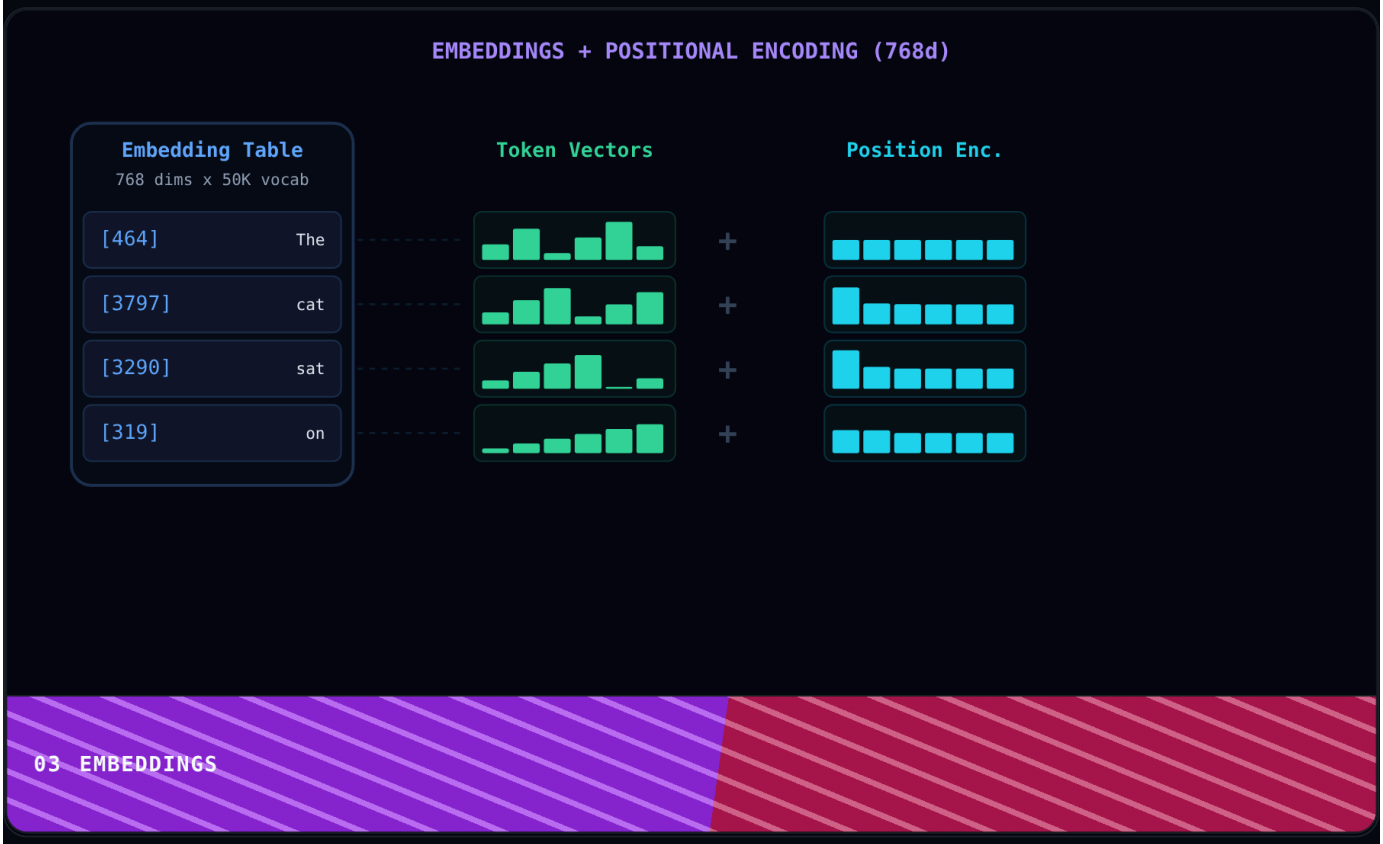
Watch the tokens appear one by one. Each colored piece is one token. Notice how common words like "The" stay whole, while less common words get split into subword chunks. Try switching the Method control in the sidebar to WordPiece and see how the splits differ slightly.

Each token maps to an integer ID from a fixed vocabulary (typically 32,000 to 100,000 entries). The model never sees text after this point. Everything downstream is pure mathematics on these integer IDs.

That is the job of tokenization: turn variable-length human text into a fixed-vocabulary sequence of integer IDs. These IDs are the input to the next stage, where each one gets converted into a rich numerical vector. Next, we will see how that conversion works through embeddings.

STAGE 3 OF 8

Embeddings



03 EMBEDDINGS

Now that we have token IDs, we need to turn each one into a vector the model can reason about. This stage shows how that happens. Remember from the Setup stage: an embedding is a list of numbers that captures meaning.

The model maintains an embedding table: a giant matrix where each row corresponds to one vocabulary entry. To embed token ID 4821, we simply look up row 4821. No computation needed, just a table lookup.

Each row is a dense vector (a list of real numbers). In GPT-3 this vector has 12,288 dimensions. In smaller models it might be 768. Try the Dimensions control in the sidebar to see how vector size changes. More dimensions means the model can encode subtler distinctions in meaning.

There is a problem: these embedding vectors know what each token means, but they have no idea where it sits in the sentence. "Dog bites man" and "Man bites dog" would look identical. We need positional information.

Positional encoding adds a unique position-dependent vector to each embedding. The original transformer used sine and cosine waves at different frequencies. Modern models learn these position vectors during training, just like the token embeddings.

The final output of this stage is a sequence of position-aware vectors. Each vector encodes both what the token means and where it sits. These vectors flow into the attention mechanism, where the model starts connecting tokens to each other. That is what we will explore next.

STAGE 4 OF 8

Self-Attention

04 SELF-ATTENTION

We just saw how each token becomes a position-aware vector. But those vectors still know nothing about each other. Self-attention is the mechanism that lets every token look at every other token and decide what is relevant.

Each input vector is projected through three learned weight matrices to produce three new vectors: a Query (Q), a Key (K), and a Value (V). Think of Q as "what am I looking for?", K as "what do I contain?", and V as "what information do I carry?"

To compute attention scores, we take the dot product of each Query with every Key. A high dot product means that query is strongly interested in that key. This creates a square attention matrix where row i, column j says how much token i attends to token j.

The raw scores are divided by the square root of the key dimension (sqrt(d_k)). Without this scaling, dot products grow large in high dimensions, pushing softmax into regions where gradients vanish. This scaling keeps training stable.

Switch the Masking control to Causal and watch the upper triangle of the matrix go dark. In language generation, each token must only attend to tokens before it. The model masks future positions with negative infinity before softmax, making those attention weights zero.

Softmax converts each row of scores into a probability distribution. Now each row sums to one. High-affinity token pairs get most of the weight. Low-affinity pairs get near zero.

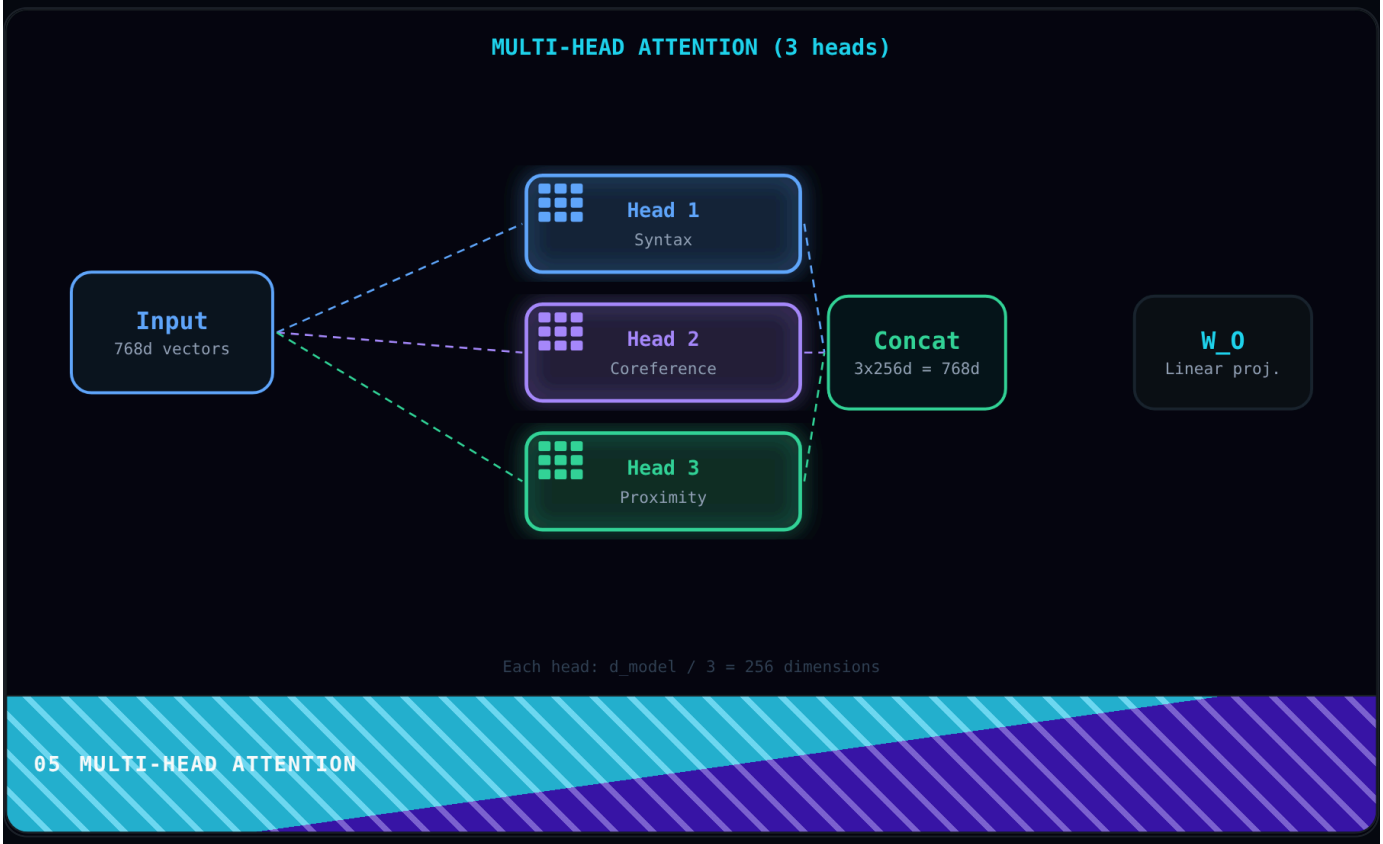
Finally, we multiply the attention weights by the Value vectors. Each output is a weighted sum of all value vectors, where the weights come from the softmax scores. Tokens that attend strongly contribute more to the output.

One attention head captures one type of relationship: maybe syntax, maybe coreference, maybe semantic similarity. But one head is not enough. Next, we will see how multiple heads run in parallel,

each learning a different pattern of attention.

STAGE 5 OF 8

Multi-Head Attention



05 MULTI-HEAD ATTENTION

A single attention head, which we just explored, captures one type of relationship between tokens. But language is rich. One head might learn syntax while another learns coreference. Multi-head attention runs several heads in parallel.

The input vectors are split into chunks, one per head. Each head gets its own Q, K, V weight matrices. These matrices are smaller since the dimension is divided by the number of heads. Use the Heads control in the sidebar to add or remove heads and watch the split change.

All heads compute attention simultaneously and independently. Head 1 might learn that adjectives attend to the nouns they modify. Head 2 might learn that verbs attend to their subjects. Head 3 might track positional proximity. Each head sees the same tokens but learns different patterns.

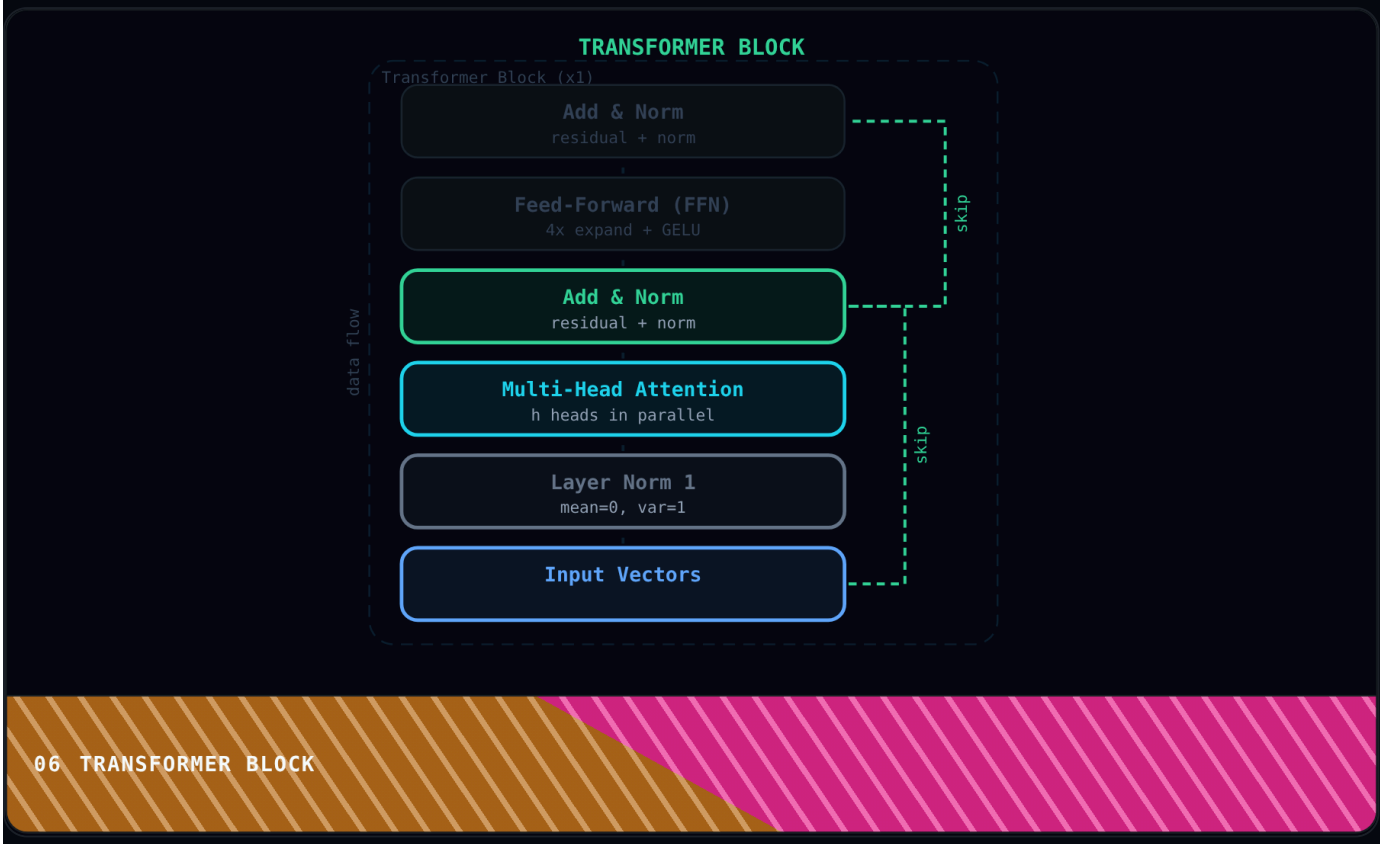
After all heads compute their outputs, the results are concatenated back into a single vector of the original dimension. If we had 3 heads each producing 256-dimensional output, concatenation gives us 768 dimensions again.

A final linear projection (a weight matrix W_O) mixes the concatenated head outputs. This lets the model combine insights from all heads into a single representation. The projection is learned, so the model decides how to weight each head contribution.

Multi-head attention is the complete attention layer. It goes into the transformer block alongside a feed-forward network and normalization layers. That full block is what we will assemble next.

STAGE 6 OF 8

Transformer Block



06 TRANSFORMER BLOCK

We now have all the pieces to build a full transformer block. Remember multi-head attention from the previous stage? It is one half of this block. The other half is a feed-forward network. Together with normalization and residual connections, they form the repeating unit of every LLM.

Input vectors enter from below. The first operation is layer normalization, which rescales the vectors to have zero mean and unit variance. This keeps the numbers in a stable range. Without it, values would drift as they pass through dozens of layers.

The normalized vectors flow into multi-head attention, the same mechanism from the previous stage. Every token attends to every other token through multiple parallel heads.

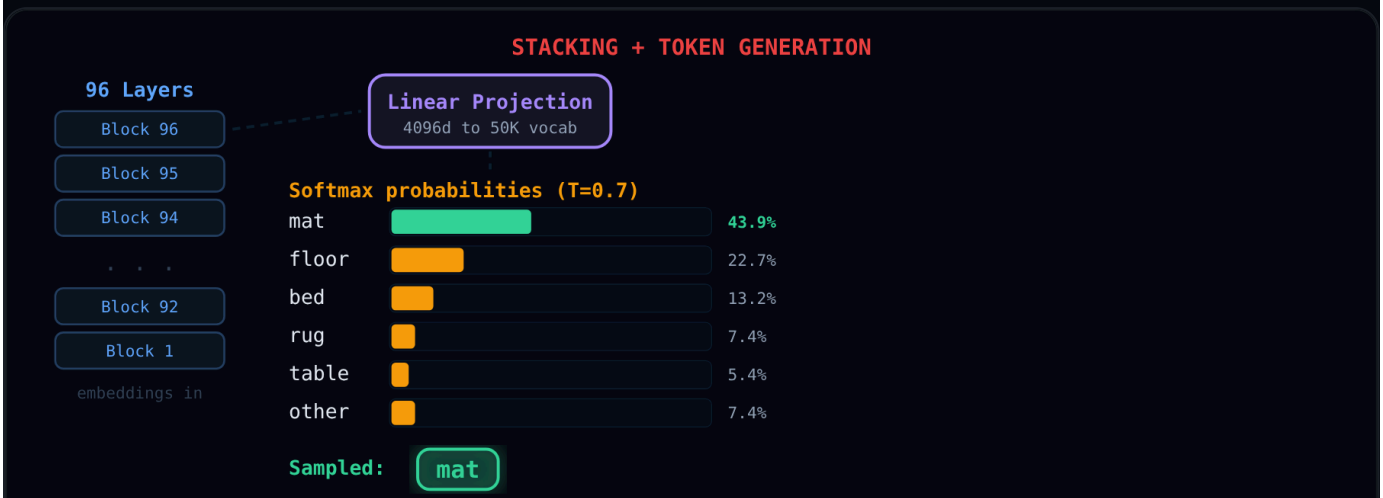
Here is the critical design choice: a residual connection adds the original input back to the attention output. Toggle the Residuals control in the sidebar to see it. This shortcut lets gradients flow directly backward through the network, making deep stacks trainable.

After another layer normalization, the vectors enter the feed-forward network (FFN). This is two linear layers with a nonlinearity (usually GELU) in between. The FFN expands the dimension by 4x, applies the nonlinearity, then projects back down. It is where the model stores factual knowledge.

A second residual connection adds the pre-FFN input back to the FFN output. The block is now complete: Norm, Attention, Add, Norm, FFN, Add. This block gets stacked many times. GPT-3 uses 96 of them. Next, we will see how stacking works and how the final output becomes a word.

STAGE 7 OF 8

Generation



07 GENERATION

We have one transformer block. A real LLM stacks dozens of them. GPT-3 has 96 layers. Each block refines the token representations further. Early layers capture syntax and local patterns. Deep layers capture abstract semantics and world knowledge.

The input token embeddings (from Stage 2) enter at the bottom and flow upward through every block in sequence. Each block applies attention and feed-forward transformations, adding its own learned refinement. The residual connections from Stage 5 keep gradients healthy across all these layers.

After the final transformer block, we have rich contextual vectors. But the model needs to predict a word, not output a vector. A linear projection maps each final vector from model dimension (e.g. 4096) to vocabulary size (e.g. 50,000). Each entry in the resulting vector is a raw score (logit) for one vocabulary token.

Softmax converts these logits into a probability distribution over the entire vocabulary. Remember softmax from the Setup? Same function, different place. The token with the highest probability is the model prediction. But we do not always pick the top one.

Temperature controls how the model samples. Low temperature sharpens the distribution, making the top token dominate. High temperature flattens it, giving unlikely tokens a real chance. Try the Temperature control and watch the probability bars change. Low values produce predictable, repetitive text. High values produce creative but sometimes incoherent text.

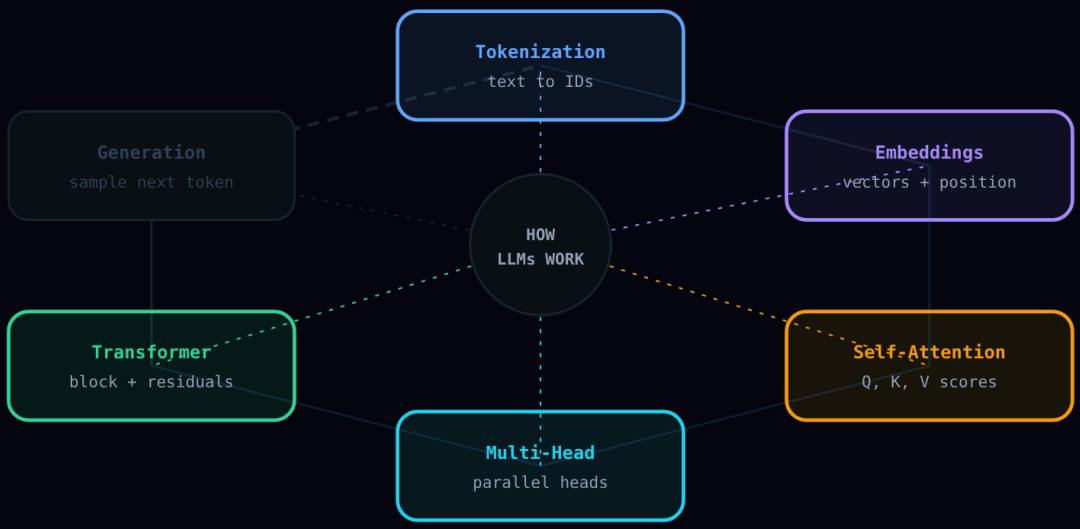
The sampled token feeds back as input to generate the next token. This is autoregressive generation: the model produces one token at a time, each conditioned on everything before it. It is the same architecture end to end. No separate "understanding" module and "generation" module. Just one stack of transformer blocks applied repeatedly.

That completes the full picture: text enters, gets tokenized, embedded, refined through attention and feed-forward layers, then decoded back into a probability over words. Now let us step back and see the

whole architecture together.

STAGE 8 OF 8

Recap



08 RECAP

We started at the very beginning: raw text. Tokenization splits that text into subword pieces and maps each piece to an integer ID. The model never sees characters or words. It sees a sequence of numbers from a fixed vocabulary.

Embeddings turn those integer IDs into dense vectors. Each vector captures meaning. Positional encoding adds location information so the model knows word order. Without position, every permutation of a sentence would look identical.

Self-attention is the core innovation. Every token computes a query, key, and value. The dot product of queries and keys produces attention scores. Softmax normalizes them. The weighted sum of values produces context-aware representations.

Multi-head attention runs several attention heads in parallel, each learning different relationships. Their outputs are concatenated and projected. This gives the model multiple "lenses" to look through at the same input.

The transformer block wraps multi-head attention with a feed-forward network, layer normalization, and residual connections. These residual shortcuts are what make it possible to stack blocks deeply without losing gradient signal.

Stacking dozens of these blocks creates the full model. A linear projection and softmax at the top decode rich contextual vectors back into word probabilities. Temperature and sampling strategies control the balance between predictability and creativity.

This is the entire architecture of a modern large language model. Text becomes tokens. Tokens become vectors. Attention connects them. Transformer blocks refine them. And a final projection turns vectors back into words. Every response you receive from an LLM follows exactly this path.