# What Are Large Language Models?

THE COMPLETE GUIDE

TROIANA STUDIO

MONOGRAPH NO. 1

# What Are Large Language Models?

AUTHOR

**Erjon Vranovci**

PUBLISHER

**Troiana Studio**

DATE

**July 2026**

# Contents

CHAPTER 01

# What Is a Large Language Model? 01

A large language model is an artificial-intelligence system trained on enormous amounts of text to predict language — and, through that prediction, to answer questions, write, summarise, translate, reason through problems, and hold a conversation.

> "At sufficient scale, predicting the next word forces a model to absorb grammar, facts, and a working model of the world."

An LLM does not know facts the way an encyclopaedia does. It has learned, from examples, which words tend to follow which other words in which contexts. At sufficient scale, learning to predict the next word forces a model to absorb grammar, facts, reasoning patterns, tone, and a working model of the world.

This guide explains how LLMs work, from the ground up. You will not need any maths. By the end you will understand what an LLM actually is, how it is built, why it behaves the way it does, which models matter in 2026, how to get better results from them, and how they are quietly rewiring the way people find information.

The short version: an LLM is a next-word-prediction machine. You give it some text — a question, an instruction, a half-finished sentence. It converts that text into numbers, runs those numbers through a very large network of learned weights, and produces a probability for every possible next word. It picks one, adds it to the text, and repeats — now predicting the word after that.

Do this a few hundred times and you have a paragraph. That is genuinely all an LLM does at the moment it answers you: predict the next piece of text, over and over, extremely fast. Everything else — the apparent intelligence, the reasoning, the code, the poetry — is an emergent consequence of doing that one task well, at a scale that was not computationally possible until recently.

CHAPTER 02

# Tokens: The Atomic Units

02

An LLM does not see letters or words the way you do. It sees tokens — the atomic units of text it was built to process. A token is often a whole short word, but longer or rarer words get split into pieces.

Tokenizer output example

`un` `believ` `able`

METRIC STANDARD

**100 tokens ≈ 75 words**

This is the standard conversion for context windows, memory allocation, and billing calculations.

Turning text into tokens is called tokenisation, and every model family has its own tokeniser. That is a small but real reason the same prompt can cost slightly different amounts, or fit differently, across ChatGPT, Claude, and Gemini.

When you hear that a model has a 200,000-token context window, that is the tokeniser units — roughly 150,000 words, or a decent-sized book, that the model can hold in view at once. A space usually belongs to the token that follows it, and punctuation is its own token.

```
UNIT STANDARDS / REF DATA
1 token ≈ 4 characters
100 tokens ≈ 75 words
200K tokens ≈ 150,000 words (full book)
```

Because the model reads token by token, it can occasionally stumble on tasks that require seeing individual letters — famously, counting how many times the letter r appears in strawberry, because it sees the word as a couple of tokens, not a string of letters.

CHAPTER 03

# Embeddings: Meaning as Geometry

03

The first thing an LLM does with each token is convert it into an embedding — a list of numbers representing the token meaning as a point in space. Meaning becomes geometry.

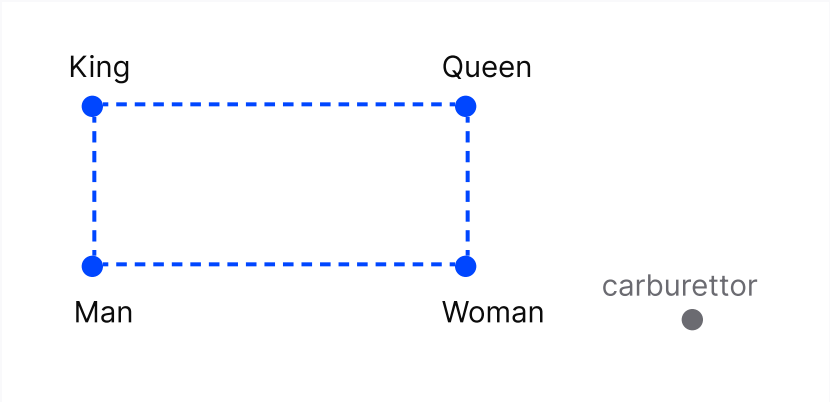
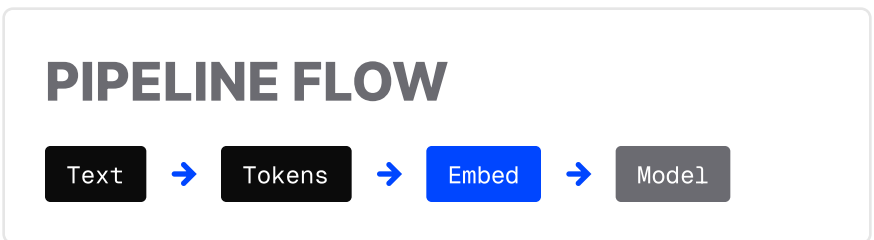



Fig. 3.1: Vector Relationships

Tokens with similar meanings end up near each other. King sits close to queen, monarch, and throne, and far from carburettor.

Embeddings are also the foundation of much of the AI tooling built around LLMs. When a system needs to find documents relevant to your question — the retrieval step in retrieval-augmented generation — it usually converts both your question and every candidate document into embeddings and looks for the ones closest together in vector space.

This is semantic search: matching on meaning rather than exact keywords, which is why it can connect how do I make my site load faster to a page titled improving Core Web Vitals even though they share almost no words.




So the pipeline so far: your text becomes tokens, and each token becomes an embedding — a vector the network can actually compute with. Now the real work begins.

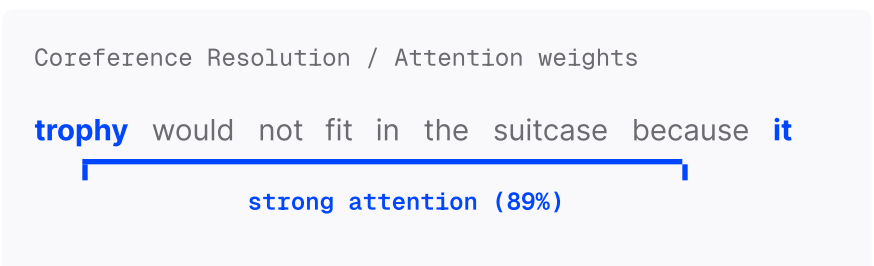
CHAPTER 04

# The Transformer Architecture

04

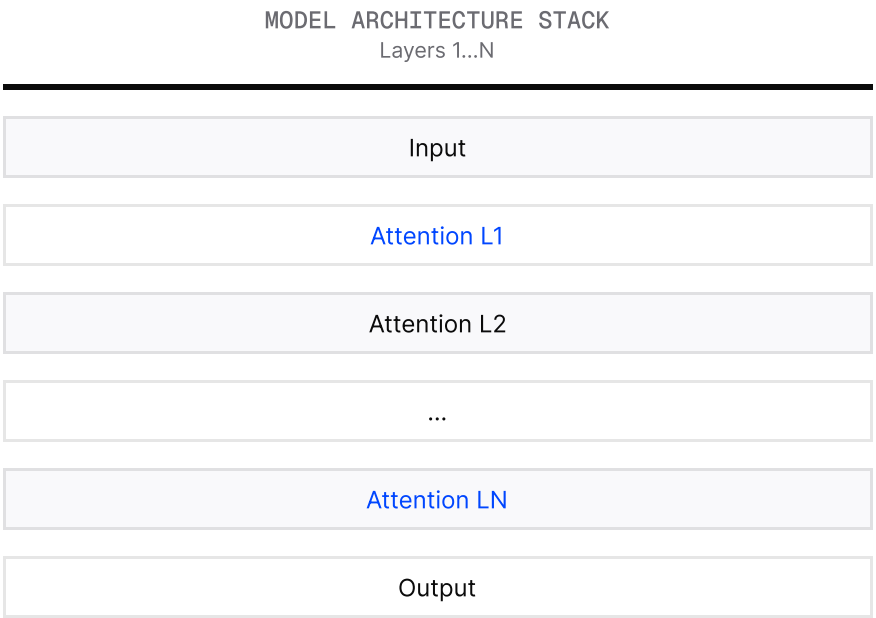
The architecture behind every modern LLM is the transformer, introduced in 2017 in a paper titled "Attention Is All You Need." Before it, models processed text sequentially, making them slow and forgetful. The transformer processes all words at once.

Its central mechanism is self-attention. For every token, the model asks: which other tokens should I pay attention to, and how much? Consider: "The trophy would not fit in the suitcase because it was too big."




The model must focus on "trophy". Change "big" to "small", and the answer flips to "suitcase". A transformer stacks dozens of attention layers with billions of parameters.

A transformer stacks many layers of attention on top of each other — dozens, sometimes over a hundred. Early layers tend to capture surface patterns like grammar; later layers capture abstract structure like intent, logic, and style.




Note: dozens to hundreds of layers

Training a model means finding the right values for its parameters. Crucially, transformers process entire passages in parallel rather than word by word, making internet-scale training possible on GPUs.

CHAPTER 05

# Training a Model

05

A newly created model is useless—billions of random parameters producing gibberish. Training transforms this noise into capability through three deliberate stages.


**1. Pretraining:** The model ingests raw Web text to master token prediction. This expensive phase yields the "base" model.

**2. Fine-Tuning:** The model trains on top of instruction pairs, learning the dialog format.

**3. Alignment:** Reinforcement learning from human feedback (RLHF) calibrates the final helpfulness, tone, and safety constraints.

Two consequences fall out of this process and matter constantly in practice. First, a model has a knowledge cutoff: it only knows about the world up to when its training data was collected.

Second, the model helpfulness is trained in, not innate. The same underlying network could be tuned to behave very differently; the assistant you talk to is a deliberately shaped product.

CAPITAL REQUIREMENTS

## Pretraining cost: $10M–$100M+ in compute

Pretraining is by far the most expensive stage. The result is a base model: fluent, knowledgeable, but undirected — it will happily continue your text without any sense that it is supposed to be helpful. Anthropic's approach, Constitutional AI, is a well-known variant that uses a written set of principles.

CHAPTER 06

# The Mechanics of Inference

06

When you send a message, the model runs in inference mode. Your entire conversation is converted into tokens and fed into the transformer network.

The model does not generate sentences whole. It computes a probability distribution over its entire vocabulary for the next single token, samples one, appends it to your prompt, and repeats the loop. This is why generated text streams out word by word.

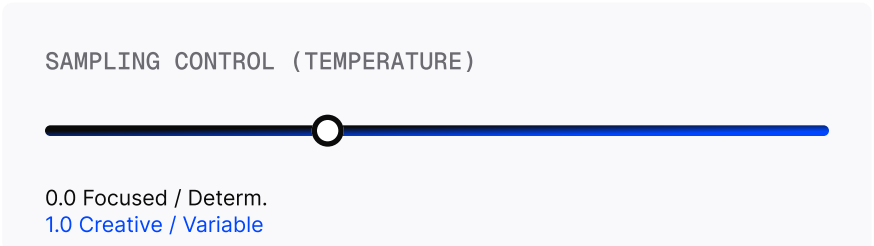
SAMPLING PARAMETER: TEMPERATURE

**Low Temp (0.1 - 0.3)**
Deterministic, precise, mathematically rigid. Best for code.

**High Temp (0.7 - 1.0)**
Creative, variable, prone to non-standard paths.

Sampling is probabilistic, so identical prompts can yield completely different outputs on successive runs. The temperature parameter decides whether the model selects only the highly predictable top tokens or occasionally takes more adventurous paths.

Since generation involves sampling, the same prompt can produce different answers each time—the model is not broken; it is drawing from a probability distribution. Setting the temperature to zero makes the output nearly deterministic.




Second, the model has no memory between separate conversations unless the application deliberately stores context. Within a single chat it remembers only because the whole transcript is re-sent with every message. This is why answers stream out word by word: you are watching the model think in real time.

CHAPTER 07

# The Context Window

07

The context window is the absolute limit of what a model can hold in active memory at one time. Everything must fit inside this budget: the developer's system instructions, reference files, preceding conversational turns, and the response currently being generated.

When a conversation overflows this memory boundary, the oldest tokens are discarded. If the system prompt gets dropped, the model completely forgets its core instructions.

WORKING MEMORY CONSTRAINTS

2019 2,048 tokens (Short article)

**2026** 1,000,000+ tokens (Entire libraries)

Despite large context windows, models can exhibit the "lost-in-the-middle" effect, giving less attention to information placed in the prompt's center.

CHAPTER 08

# Why Models Hallucinate 08

Large language models will generate factual errors with absolute authority. They will confidently invent academic citations, legal cases, medical metrics, or software dependencies that simply do not exist.

This behavior is not a bug; it is a fundamental property of next-token prediction. A model is trained to be plausible, not factual. It operates without a reality-checking loop or internal metadata tags separating actual truth from persuasive fiction.

CRITICAL CONCEPT

"Fluency and truth are heavily correlated in training datasets—but structurally, they remain completely separate phenomena."

Neural networks optimize for pattern prediction, not factual verification. As a result, plausible falsehoods can be produced with confidence.

CHAPTER 09

# Retrieval-Augmented Generation

09

To bypass the limits of pretraining, developers use Retrieval-Augmented Generation (RAG). RAG links an LLM to external data stores—such as a proprietary company wiki or a real-time web index.

RAG Cycle Architecture

**User Query** → Retrieve Documents

Context Injection → **Grounded Answer**

When a user asks a question, the system queries the database first, pulls out the most relevant paragraphs, and inserts them directly into the LLM's context window.

The model then reads the question and retrieved references together, writing an answer explicitly grounded in facts. This provides private data access without expensive retraining.

CHAPTER 09 (CONTINUED)

# Connecting to Real-time Worlds

Retrieval-Augmented Generation (RAG) acts as an interface between frozen parametric weights and the dynamic, open web. Instead of relying solely on pre-existing patterns, RAG pulls directly from live websites, corporate databases, or private repositories before compiling a response.

**RAG PROCESS FLOW**

Query → **Search Index** → Augmented Prompt

By grounding the generation in verified reference documents, the system dramatically reduces hallucination. Because serious business applications demand absolute accuracy and proprietary context, RAG has transitioned from a specialized optimization to a core architecture of industrial AI.

CHAPTER 10

# Tool Use and Agents

10

An LLM on its own can only produce text. But text can be an instruction — and that turns a language model into something that takes actions. This is tool use.

You describe available tools — web search, calculator, database, code interpreter — and when the model decides a tool would help, it emits a structured request. The software runs that tool and feeds the result back.

> "Chain several steps together and you get an agent — an LLM working toward a goal over multiple steps, deciding which tools to use."

This pattern is the foundation of coding assistants, research assistants, and workflow automations, where the model adjusts its plan based on the feedback it receives from the tools it operates.

CHAPTER 11

# Beyond Text: Multimodality 11

Modern models are no longer blind. They process images, audio, video, and code alongside standard text, utilizing unified transformer architectures. By converting non-textual data into the exact same token spaces, the neural network learns to relate visual structures directly to language.

Text Image Audio Code

You can feed a model complex diagrams, maps, or UI screenshots, asking it to write operational code based on raw pixels. Voice inputs bypass traditional speech-to-text models, converting audio frequencies directly into cognitive concepts. However, core limitations like context window fatigue still apply.

CHAPTER 11

# The Major Models in 2026

11

| MODEL | MAKER | Known For |
|---|---|---|
| **Claude** | Anthropic | Strong reasoning, writing, coding, safety-first. |
| **GPT** | OpenAI | Brought LLMs mainstream, broad capability. |
| **Gemini** | Google | Large context, native multimodal, Google integration. |
| **Llama** | Meta | Leading open-weight, free to run privately. |
| **Mistral** | Mistral AI | Efficient European open-weight models. |
| **DeepSeek** | DeepSeek | Strong reasoning at low training cost. |

CHAPTER 12

# Strengths and Weaknesses 12

## STRENGTHS

**Writing & Editing**
Fluent prose generation.

**Teaching**
Explaining complex topics.

**Coding**
Generating & debugging logic.

**Extraction**
Structuring messy data.

**Ideation**
Divergent brainstorming.

## WEAKNESSES

**Precision**
Hallucinations & inaccuracies.

**Currency**
Knowledge cut-off limits.

**Reasoning**
Logical failures in math.

**Bias**
Mirroring training data flaws.

**Explainability**
The 'Black Box' problem.

CHAPTER 13

# How to Write a Good Prompt

13

**01** **Be specific.**
Define exactly what the output should look like.

**02** **Give context.**
Explain the 'why' and the target audience.

**03** **Show, don't just tell.**
Provide few-shot examples for style matching.

**04** **Ask for structure.**
Request JSON, tables, or markdown lists.

**05** **Reason first.**

Ask the model to think step-by-step before answering.

**06** **Iterate.**
Treat the first result as a conversation starter.

> *"You are not typing a search query. You are briefing a capable but literal-minded collaborator."*

CHAPTER 14

# LLMs and Search

14

The most far-reaching effect of LLMs is what they are doing to how people find information. For twenty-five years, searching meant keywords and links.

Now people ask questions and get synthesised answers from an LLM. If the answer engine does not cite your page, you are invisible. This is Generative Engine Optimization (GEO).

**OPTIMIZATION HIERARCHY**

1. Credibility & Authority
2. Clean Structural Data
3. Self-Contained Facts
4. Direct Human Value

What makes content easy for an LLM to cite is what makes it good for humans: clear answers, facts, and credibility.

CHAPTER 15

# Choosing the Right Model 15

System architects must balance parameters, privacy, latency, and operational cost when matching models to production workflows.

**DECISION MATRIX**

| | |
|---|---|
| **Need Max Capability** | Closed API |
| **Strict Privacy Needs** | Open-weight |
| **High-Volume / Simple** | Small Model |
| **Complex Reasoning** | Large Model |

Four key operational pillars: Match tier to task, decide open vs closed, weigh the developer ecosystem, and aggressively run tests on actual workload samples.

# Glossary

**LLM**
Large-scale transformer model predicting text.

**TOKEN**
Atomic segment of text processed by AI.

**EMBEDDING**
Mathematical vector representing word meaning.

**TRANSFORMER**
Parallel-processing attention architecture.

**ATTENTION**
Weighted relation scores between tokens.

**PARAMETERS**
Adjustable numeric neural network weights.

**PRETRAINING**
Initial mass prediction training stage.

**FINE-TUNING**
Task adaptation using narrow training data.

**RLHF**
Reinforcement alignment from human feedback.

**INFERENCE**
Execution of network logic to generate text.

**TEMPERATURE**
Randomness coefficient in word selection.

**CONTEXT WINDOW**
Maximum memory range of the system.

**HALLUCINATION**
Fluent sounding but false information.

**RAG**
External document search enhancement.

**AGENT**
Autonomous workflow execution loop.

**GEO**
Generative engine optimization methods.

# Common Questions

### What is an LLM in simple terms?

An advanced auto-complete program trained on millions of texts to recognize contextual language relationships.

### How is it different from a search engine?

Search engines locate existing documents via keywords; LLMs generate completely new paragraphs word by word.

### Why do LLMs make things up?

Because they calculate probability, not truth. They prioritize grammatical fluency over factual correctness.

### What is the difference between ChatGPT, Claude, and Gemini?

These are applications powered by different underlying models developed by OpenAI, Anthropic, and Google respectively.

# Common Questions

**What is a token?**

A computational unit of text—usually a chunk of 3 or 4 letters—used to translate natural language into statistics.

**What is a context window?**

The computational boundary of memory, limiting how much input text a model can hold in focus at one time.

**Do I need to know how LLMs work?**

Understanding tokenization and attention boundaries prevents common user errors and optimizes pipeline design.

**How are LLMs changing SEO?**

By routing traditional keyword searches into Generative Engine Optimization formats, rewarding direct structured data.

# troiana studio

troiana.net

Design and development studio.