APPLIED AI FOR DEVELOPERS

LLM Fundamentals I

From Text to Prediction

Understand how Large Language Models turn text into intelligent responses.

Goal of this module

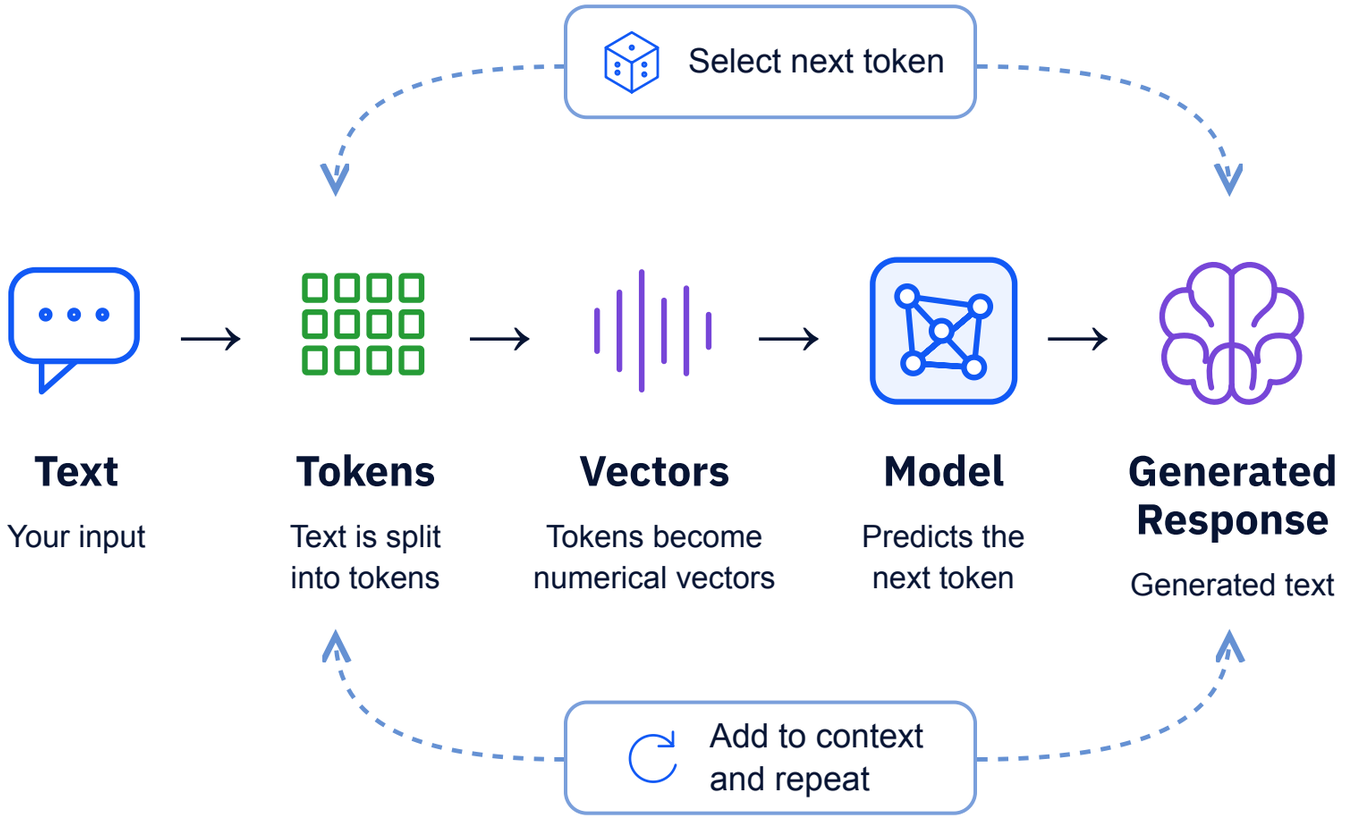
Understand the fundamental principles behind LLMs – from text input to next-token prediction.

For Developers

Build the mental model you need to work effectively with LLMs in real applications.

Next

LLM Fundamentals II – Inside the Model: Transformer, Attention, Context and Limits.



An LLM generates text by repeatedly predicting **the next token.**

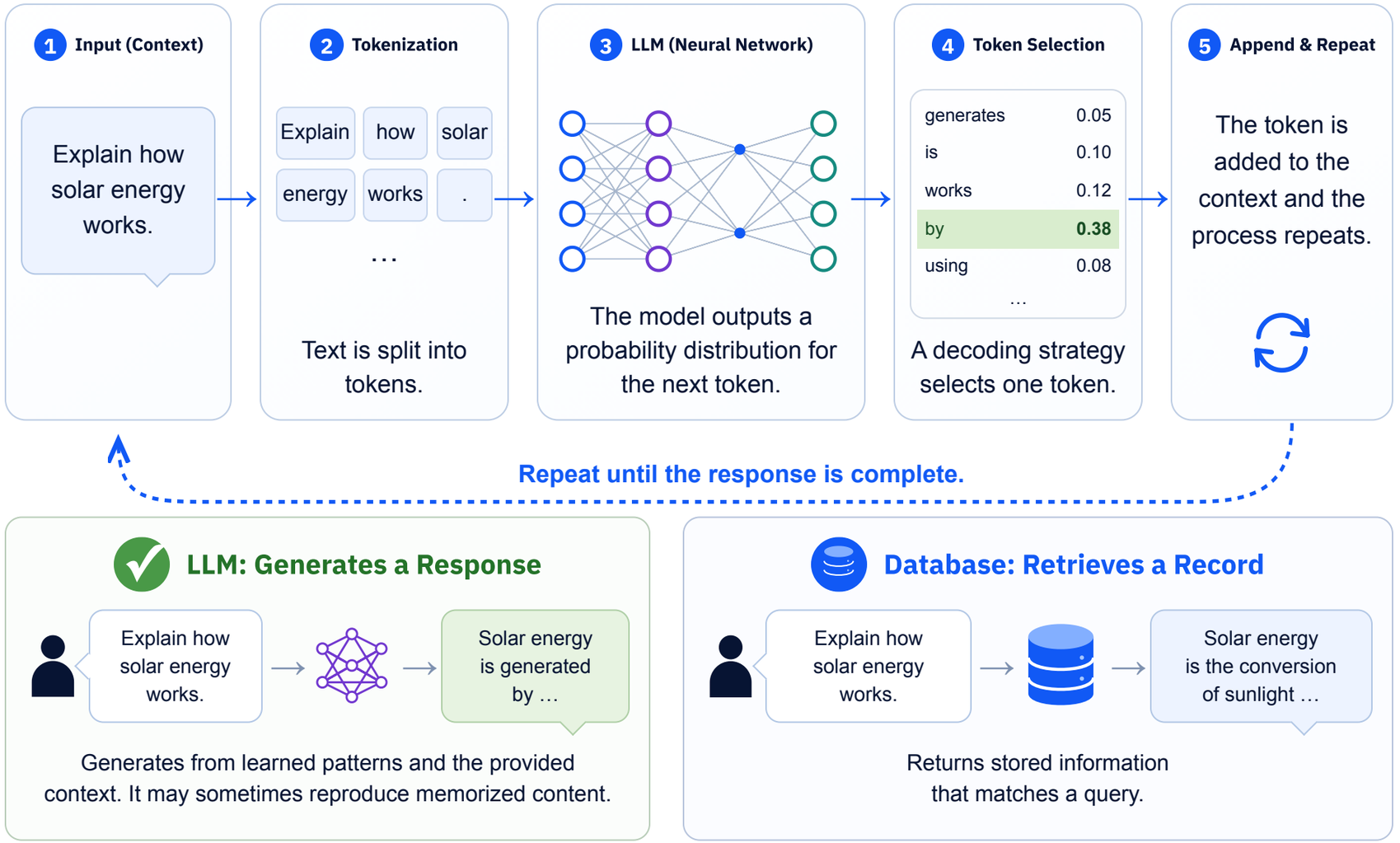
What Is a Large Language Model?

An **LLM (Large Language Model)** is a neural network trained on large amounts of text to process and generate language.

Given a context, it predicts a probability distribution for the next token. A decoding strategy selects one token, which is added to the context. Repeating this process produces answers, summaries, translations, code, and other content.

An LLM **does not normally retrieve an answer as an exact database record.** It generates a response from learned patterns, although it may sometimes reproduce memorized training content.

HOW AN LLM WORKS (AT A HIGH LEVEL)



In one sentence:
An LLM generates language by repeatedly predicting a probability distribution for the next token and selecting one token from it.

An LLM Alone Is Not a Complete AI Application

An LLM is the language engine.
A complete AI application combines the model with data, tools, workflows, security, and a user experience to solve real problems.

LLM (Large Language Model)



Language Model

Processes context and predicts a probability distribution for the next token.

+

Complete AI Application



Complete Solution

Combines the model with other components to deliver value to users.

What a Complete AI Application Typically Includes



Data & Knowledge

Documents, databases, APIs, and domain knowledge.



Retrieval & Search

Find relevant information using search, indexing, or RAG.



Prompting & Orchestration

Coordinate instructions, workflows, and model calls.



Tools & Integrations

Connect APIs, business systems, calculators, and services.



Guardrails & Security

Manage privacy, permissions, validation, and compliance.



UI / UX

Create a useful experience across web, mobile, and chat.



Evaluation & Monitoring

Measure quality, cost, latency, usage, and feedback.



Key takeaway:
An LLM is one component. Real value comes from combining it with data, tools, processes, security, and user experience to solve real problems.

The Core Idea:
Predict the Next Token

At the heart of every LLM is a simple idea:
given some text, predict what comes next.

The model doesn't know the full answer in advance.
It generates text one token at a time.

Example

Prompt (context):

The future of **renewable energy** is →

Possible next tokens (with probabilities)

bright	**0.68**
important	0.17
exciting	0.06
uncertain	0.03
…	…

The model outputs a probability distribution over the entire vocabulary.
It is not choosing the "right" answer – it is predicting what is **most likely.**

How text is generated



Key takeaway:
LLMs generate text by repeatedly predicting the next token and extending the context.

APPLIED AI FOR DEVELOPERS

From Text to Tokens

LLMs process text by breaking it into tokens. Depending on the tokenizer, tokens may represent **whole words, parts of words, punctuation, whitespace,** or other byte sequences.

LLMs do not process text like humans.
They process sequences of tokens.

Examples of Tokenization

Whole Words

Some common words may form a single token.

the | future | energy

Parts of Words

Rare or complex words may be split into subword tokens.

un | believ | able | micro | grid

The exact split and notation depend on the tokenizer.

Punctuation

Punctuation may form separate tokens or be combined with nearby text.

, | . | ! | ? | ; | : | (|) | “

Whitespace & Special Characters

Spaces and line breaks may be separate tokens or encoded as part of another token.

space | \n | \t

(illustrative)

The Tokenization Process (Simplified)

1 Input Text

Raw text from a user or document.

The future of renewable energy is bright.

2 Split into Tokens

The tokenizer splits or groups the text according to its vocabulary and rules.

The | future | of | renewable | energy | is | bright | .

3 Token IDs

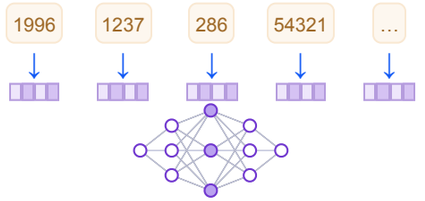
Each token maps to an ID within that tokenizer’s vocabulary.

1996 | 1237 | 286 | 54321 | 12589 | 318 | 4567 | 13

Illustrative IDs

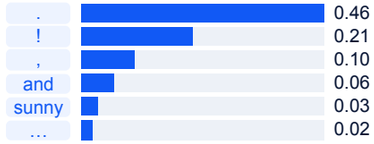
4 Embedding Lookup

Token IDs are converted to learned vectors before entering the Transformer.



5 Model Processing

The Transformer processes the vectors and predicts what may come next.



Next-token probabilities

Key takeaway:
Text → Tokens → Token IDs → Embeddings → Transformer. Tokenization converts text into units that an LLM can process and generate.

APPLIED AI FOR DEVELOPERS

Tokenization in Practice

The same sentence may be tokenized differently by different models and tokenizer versions.

There is no universal “correct” tokenization.
It depends on the **tokenizer’s vocabulary and rules.**

Example Sentence

> *The future of renewable energy is bright.*

Illustrative Token Splits

Tokenizer A
Word-oriented vocabulary

The	future	of	renewable	energy	is	bright	.

Illustrative: 8 tokens

Tokenizer B
Different vocabulary

_The	_future	_of	_renew	able	_energy	_is	_bright	_.

Illustrative: 9 tokens

Tokenizer C
Smaller subword units

_The	_fu	ture	_of	_rene	wa	ble	_en	ergy	_is	_bri	ght	_.	

Illustrative: 13 tokens

These examples show the principle only. Exact tokens and IDs must be obtained from the actual tokenizer version.

What’s Happening?

- Different vocabularies produce different token splits.
- The same sentence may use a different number of tokens.
- Token count affects context usage, cost, and latency.
- Always test with the exact model and tokenizer version you deploy.

What Affects Tokenization?

Vocabulary
Determines which text units are available.

Algorithm
BPE, Unigram, WordPiece, and related methods split text differently.

Language
Languages and writing systems may tokenize differently.

Whitespace & Rules
Spaces, punctuation, capitalization, and special tokens matter.

Model & Version
Tokenization may change across model families or versions.

Try It Yourself

Use the tokenizer provided for your selected model.

```
tokenizer = load_tokenizer(model)
tokens = tokenizer.encode(text)
print(tokens)
print(len(tokens))
```

- ✓ Token IDs
- ✓ Token count
- ✓ Decoded tokens

Key takeaway: **Tokenization is model-dependent. Test the exact tokenizer you deploy to manage context limits, estimate costs, and avoid unexpected behavior.**

Why Tokens Matter for Developers

Tokens are not just a technical detail.
They directly impact the **behavior, performance,** and **cost** of your AI applications.

Every token you send or receive has an impact.

Understand tokens. Build better.

How Tokens Impact Your Application

1 Context Limits

Tokens determine how much the model can “see” at once.



Used: **7,500 tokens**
Limit: **8,192 tokens**
Illustrative context budget

Why it matters:
Exceeding the limit may reject a request or truncate context.

2 API Costs

LLM API usage is commonly priced by input and output tokens.



Why it matters:
More tokens can mean higher cost. Optimize what you send and generate.

3 Latency

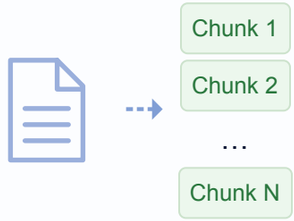
More tokens usually mean more computation and longer responses.



Why it matters:
Longer prompts and outputs can increase response time for your users.

4 RAG Chunking

Split your data into token-aware chunks for retrieval.



Why it matters:
Chunk size (in tokens) affects retrieval quality and relevance.

Tokens Are a Real Resource

Input Tokens
Tokens you send in your prompt (system + user + context).

Output Tokens
Tokens generated by the model in its response.

Total Tokens
Input + Output = total usage. Check model-specific limits.

Illustrative Example

Prompt: 2,000 tokens
Response: 1,000 tokens
Total: 3,000 tokens

Cost = 2 × input rate + 1 × output rate
Rates per 1K tokens

Best Practices for Developers

Be concise
Remove unnecessary words, repetition, and verbose context.

Summarize
Summarize long content before sending it.

Filter early
Only include relevant information in the prompt.

Control output
Limit maximum output tokens when possible.

Monitor & optimize
Track token usage, costs, and latency in production.

Key takeaway:
Tokens are a limited and valuable resource. Designing with tokens in mind leads to faster, cheaper, and more reliable AI applications.

APPLIED AI FOR DEVELOPERS

From Tokens to Numbers

LLMs work with numbers, not text.
Every token is mapped to a unique token ID (a number).

The model never sees words – it sees sequences of numbers.
Numbers in, numbers out.

The Journey: Text → Tokens → Numbers

1 Text (Human Language)

The future of renewable energy is bright.

Human writes or provides text

2 Tokens

The | future | of | renewable | energy | is | bright | . | (end)

Text is split into tokens (words, parts, punctuation)

3 Token IDs (Numbers)

1996 | 1237 | 286 | 54321 | 12589 | 318 | 4567 | 13 | 2 (end)

Each token is mapped to a unique number (ID)

Illustrative tokens and IDs; special tokens depend on the tokenizer.

How It Works

1 Input Text

You provide text (prompt).

2 Tokenization

Text is split into tokens.

3 Token to ID Mapping

Token	ID
The	1996
future	1237
of	286
…	…

Each token is mapped to a number.

4 Model Processing

The model processes numerical representations.

5 Output: IDs → Tokens → Text

54321 | 12589 | 318 | … | 2 (end)

renewable energy is …

Output IDs are converted back to tokens and text.

Why This Matters

Efficient Computation
Numbers are fast for computers to process.

Embeddings & Vectors
Token IDs are used to look up learned vector representations.

Foundation of Generation
The model predicts the next token ID, which is converted back to text.

Deterministic Mapping
The same token maps to the same ID for a given tokenizer vocabulary.

Key takeaway:
Tokens are the building blocks of language, but LLMs operate on numbers. Understanding this bridge is key to building efficient and reliable AI applications.

APPLIED AI FOR DEVELOPERS

From Token IDs to Vectors

Token IDs are symbolic indices. Before the Transformer can process them, each ID is mapped to a learned vector called a token embedding.

Token embeddings provide the **starting representation.**
Context-dependent representations emerge inside the Transformer.

1 Token IDs

Each ID identifies a token within one tokenizer's vocabulary.

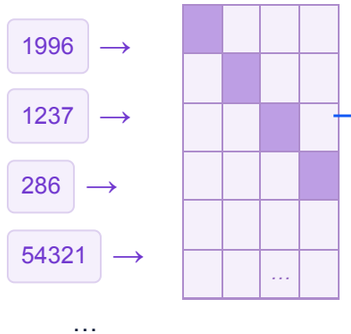
Illustrative

1996
1237
286
54321
…

Each ID represents a token.

2 Embedding Lookup

Each ID selects one row from the learned embedding matrix.



Embedding matrix
Vocabulary size × vector dimension

3 Token Embeddings

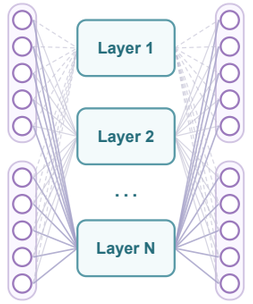
Each token starts as a dense vector.

1996 → [0.21 −0.47 0.18 … 0.73]
1237 → [−0.11 0.62 −0.33 … 0.08]
286 → [0.74 0.05 −0.19 … −0.41]
54321 → [0.33 −0.22 0.91 … −0.12]
…

Learned during training.

4 Transformer Layers

Attention and other operations combine information across token positions.



Information mixes across tokens and layers.

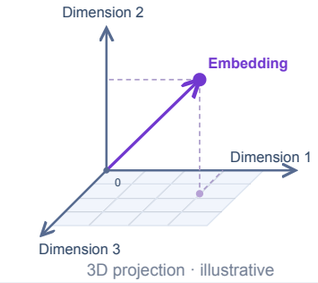
5 Contextual Representations

The vector for each position now depends on the surrounding context.

1996 → [0.91 −0.13 0.67 … 0.22]
1237 → [−0.43 0.81 −0.26 … 0.12]
286 → [0.11 0.64 −0.72 … −0.36]
54321 → [0.57 −0.08 0.31 … −0.63]
…

Context influences each position's representation.

What Is an Embedding?



3D projection · illustrative

A learned point in a high-dimensional numerical space.

- Dimensions are learned, not hand-designed.
- Related usage patterns may produce related representations.
- Initial token embeddings are context-independent.

Same Token, Different Context

Context	Token ID	Contextual output
"river bank"	8123	[0.82 −0.31 … 0.19]
"bank account"	8123	[−0.22 0.73 … 0.64]

Same token ID and initial embedding.
The contextual representation changes.

Illustrative ID and vector values.

Key Distinctions

- **Token ID** — symbolic index
- **Token embedding** — learned starting vector
- **Contextual representation** — output of Transformer layers
- **Internal computation** — numerical vector operations

Key takeaway:
Token IDs are mapped to learned embeddings. Transformer layers then turn them into context-dependent representations.

What Goes Into the Model?

Before generation, messages and relevant context are serialized, tokenized, and processed by the model.

Generation parameters control how the output is selected.

The model can use only information represented in its learned parameters or provided in the current context.

The Ingredients of Model Context

1 System / Developer Instructions

Define behavior, constraints, and priorities.

Example
Answer clearly and concisely.

+

2 User Prompt

The current request or question.

Example
Explain how solar panels work.

+

3 Context / Retrieved Data

Relevant documents, records, or tool results supplied for this request.

Example
Selected content from a technical document.

+

4 Conversation History

Previous messages included to maintain continuity.

Example
Earlier discussion about installation costs.

Serialized Messages → Tokens → Transformer → Next Token Selection

Only included content consumes context-window tokens.

Generation Parameters

Control decoding — not part of the tokenized prompt

- **Temperature** — Reshapes token probabilities.
- **Top-p / Top-k** — Restrict candidate tokens.
- **Max output tokens** — Limits response length.
- **Stop conditions** — End generation.

How It Flows

Messages & Context → Serialization → Tokenization → Transformer → Token Probabilities → Decoding → Response

Why It Matters

- **Relevance** — Include information needed for the task.
- **Priority** — Instruction roles influence behavior.
- **Reliability** — Relevant context can reduce unsupported answers, but does not guarantee correctness.
- **Efficiency** — Unnecessary context increases tokens, latency, and cost.

Key takeaway:
Messages and supplied data form the model context. Generation parameters separately control how the next token is selected.

APPLIED AI FOR DEVELOPERS

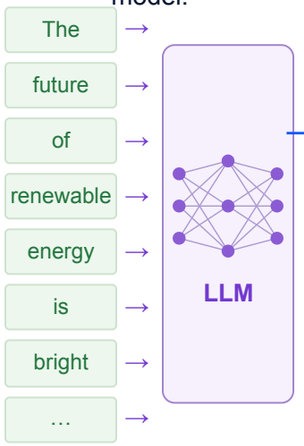
From the Model to Token Probabilities

After processing the input, the model predicts what tokens are most likely to come next. It outputs a **probability distribution** over the entire vocabulary.

The model assigns a probability to every possible token. **Decoding selects the next token.**

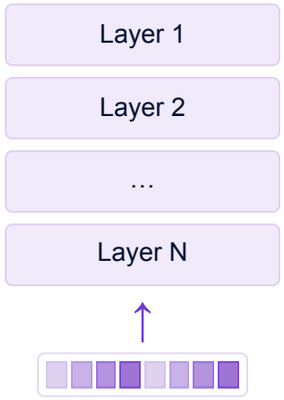
1 Input to the Model

Tokenized input is converted to embeddings and passed through the model.



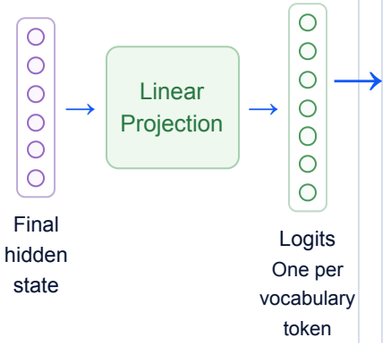
2 Model Processing

The model processes the sequence using attention and layers.



3 Next-Token Prediction

The final hidden state is projected to vocabulary scores for the next token.



4 Logits to Probabilities

Logits are converted to probabilities using Softmax.

Softmax

Converts logits to probabilities

$p_i = \exp(z_i) / \Sigma_j \exp(z_j)$



$\Sigma\, p_i = 1$

5 Probability Distribution

A probability for every token in the vocabulary. Only a few have high probability.

Token	ID	Probability
bright	5022	0.327
promising	8121	0.128
clean	3416	0.097
exciting	9910	0.056
the	1996	0.031
…	…	…
zzzzzzz	250111	0.000001

Illustrative subset of a 250,112-token vocabulary.

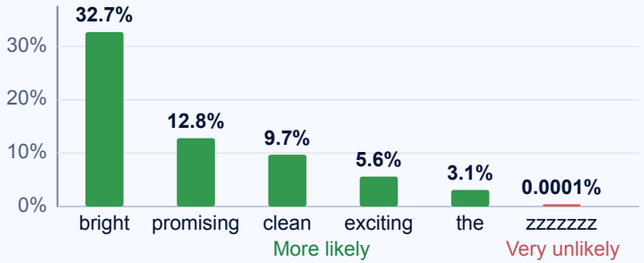
How Probabilities Look

Example context:

The future of renewable energy is

Illustrative probabilities

Probability distribution over next tokens



Top-k Example (k = 5)

Keep the five most likely tokens:

1.	bright	0.327
2.	promising	0.128
3.	clean	0.097
4.	exciting	0.056
5.	the	0.031

Divide by **0.639** to renormalize, then sample.



Why This Matters

- **The model predicts, it doesn't know.** Probabilities reflect learned patterns.
- **Generation is a choice.** Decoding selects or samples a token.
- **Developers control the behavior.** Temperature, top-k and top-p shape selection.
- **Probabilities power generation.** Used in chat, summaries and code completion.

Key takeaway:
The model turns your input into a probability distribution over all possible next tokens. These probabilities are the foundation of every generation step.

APPLIED AI FOR DEVELOPERS

Generating Text – One Token at a Time

LLMs generate text **autoregressively**:
They predict one token, add it to the sequence, and repeat until a stopping condition is met.

The model builds the response step by step, **one token at a time.**

How a Single Step Works

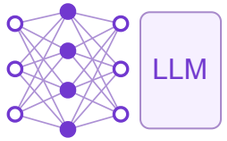
1 Current Input (So Far)

The sequence so far is the model's current input.

The future of renewable energy is bright …

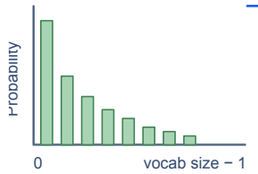
2 Model Processes the Input

The model converts tokens to embeddings and processes them.



3 Predict Next Token

The model predicts a probability distribution over the vocabulary.



4 Select Next Token

Choose a token using greedy decoding, top-k, or another strategy.

.

Selected token (end of sentence)

5 Append to Sequence

Add the selected token to the sequence.

The future of renewable energy is bright . …

6 Repeat or Stop

Repeat with the new sequence until a stopping condition is met.

Continue generating the next token.

The Autoregressive Generation Loop

Start

Prompt

"The future of renewable energy is"

1. Process

Process the current input.

2. Predict

Predict token probabilities.

3. Select

Choose the next token.

4. Append

Append the selected token.

Generated Text

The future of renewable energy is **bright.**

Repeat until a stopping condition:

- End-of-sequence token (EOS)
- Output limit reached · Stop sequence encountered

Common Decoding Strategies

Greedy (Top-1)

Always pick the most probable token.

Top-k Sampling

Sample from the k most likely tokens.

Nucleus (Top-p) Sampling

Sample from the smallest set with total probability ≥ p.

What Controls Generation?

Temperature
Higher: more diverse.
Lower: more focused.

Top-k / Top-p
Restrict the candidate tokens considered.

Max Output Tokens
Limit the generated response length.

Stop Sequences
Tokens or strings that stop generation.

Key takeaway:
Generation is a loop: process → predict → select → append → repeat. Each selected token becomes part of the input for the next step.

APPLIED AI FOR DEVELOPERS

How Is the Next Token Selected?

The model assigns probabilities to tokens. A **decoding strategy** turns this distribution into a single next token.

Different strategies can produce **different results.**

From Probabilities to One Token

1 Model Output

The model produces a probability distribution over the vocabulary.

Token	Probability
bright	0.327
promising	0.128
clean	0.097
exciting	0.056
the	0.031
…	…
zzzzzzz	0.000001

2 Apply Decoding Strategy

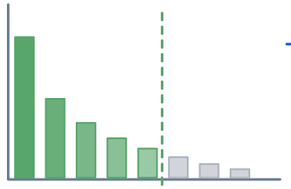
The strategy turns probabilities into a token choice.

Strategy examples

- Greedy (Top-1)
- Top-k sampling
- Nucleus (Top-p)
- Temperature
- Repetition penalty

3 Filter / Adjust

Adjust probabilities or filter candidates using the chosen parameters.



Example: **Top-k** keeps only the top 5 tokens.

4 Sample or Pick

Select one token from the candidate distribution.



Deterministic with greedy decoding, or random with sampling.

5 Next Token

Use the selected token as the next token in the sequence.

Selected Token

bright

Append this token and continue generating.

Decoding Strategies at a Glance

Strategy	How It Works	Parameters	Behavior	Useful For
Greedy (Top-1)	Pick the token with the highest probability.	—	Deterministic; can be repetitive.	Focused output and consistency.
Top-k Sampling	Sample from the k most likely tokens.	k: 10, 20, 50	Controlled randomness; limits the tail.	Balanced diversity and coherence.
Nucleus (Top-p)	Sample from the smallest set with total probability ≥ p.	p: 0.8, 0.9, 0.95	Adaptive candidate-set size.	Open-ended text, chat and stories.
Temperature	Scale logits before Softmax.	T: 0.2–2.0	Lower: more focused. Higher: more varied.	Tune output diversity.
Repetition Penalty	Adjust scores of tokens already generated.	e.g. 1.1–2.0	Discourages repetition and loops.	Longer text; implementation varies.

How Parameters Affect the Output

Temperature (T)

Lower (0.2) ⟷ Higher (1.5)

More focused — The future of renewable energy is **bright.** … Renewable energy is full of possibilities. — More creative, more varied

Top-k (k)

Smaller k (e.g. 5) ⟷ Larger k (e.g. 50)

More focused, less diverse — … — More diverse, more random

Top-p (p)

Lower p (e.g. 0.8) ⟷ Higher p (e.g. 0.95+)

More focused, smaller set — … — More diverse, larger set

Repetition Penalty

Lower penalty (1.0) ⟷ Higher penalty (1.5+)

More repetition possible — the the the the the … — … — the future of renewable energy is bright … — Fewer repeats

Key takeaway:
The model gives probabilities; decoding turns them into a decision. Tune consistency, diversity and creativity — and evaluate factual accuracy separately.

APPLIED AI FOR DEVELOPERS

What Does the Model Actually Know?

An LLM is **not a verified knowledge database.**

Its responses come from learned parameters, the current context, and — when connected — external tools or retrieval.

Fluent output can still be **incomplete, outdated, or wrong.**

Three Sources That Shape a Response

Learned Parameters

Patterns learned during training.

- Language and code patterns
- Associations and task strategies
- Some facts, imperfectly represented

Current Context

Information included in this request.

- Instructions and conversation
- Documents or retrieved passages
- Examples and tool results

External Systems

Optional sources accessed via tools.

- Search, databases and APIs
- Current or company-specific data
- Calculators and business systems

Generated Response

Combines learned patterns and supplied context to predict tokens.

What Is Not Guaranteed

- **Truth** — Plausible answers may be wrong.
- **Completeness** — Relevant information may be missing.
- **Freshness** — Learned parameters are not continuously updated.
- **Source fidelity** — The model may misread or ignore supplied evidence.
- **Consistency** — Outputs may vary across runs.

How to Build Reliable Applications

- Retrieve relevant and authorized information.
- Use tools for calculations and live data.
- Ask for citations or evidence where appropriate.
- Validate important claims and outputs.
- Apply permissions, monitoring and evaluation.

Retrieval-Augmented Generation (RAG)

User Question → Retrieval → Selected Passages → LLM Context → Generated Answer

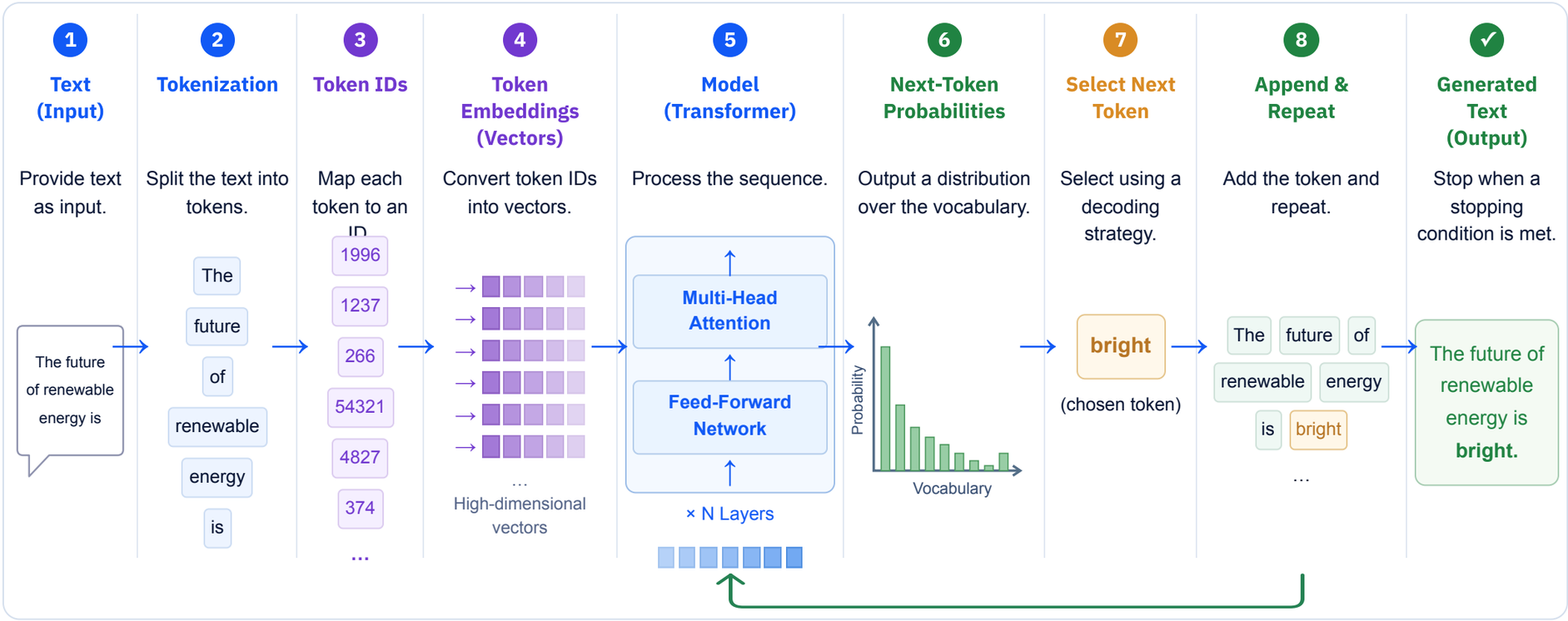
RAG supplies evidence to the context. It does not automatically make an answer correct or up to date.

Key takeaway:
An LLM contains learned statistical representations, not a verified fact store. Add relevant context and tools — and verify important results.

APPLIED AI FOR DEVELOPERS

The Complete Picture

From your text to the model's response — every step explained.



The Big Idea

LLMs turn text into tokens, process them as vectors, predict the next token, and generate text **autoregressively.**

What Happens Inside the Model?

- Attention focuses on relevant tokens.
- Layers build contextual representations.
- The final output scores every possible next token.

Key Concepts Recap

- **Tokens:** Small pieces of text.
- **Embeddings:** Vectors representing tokens.
- **Transformer:** Neural network that processes context.
- **Autoregressive:** One token at a time.

Why This Matters for Developers

- Understand model behavior.
- Use parameters more effectively (temperature, top-k, etc.).
- Debug and evaluate responses.
- Build reliable, efficient AI applications.

The Core Principle

LLMs predict the next token from **learned patterns and the current context** — one step at a time.

This simple loop powers highly capable systems.

Key takeaway:
The complete flow is a loop: tokenize → embed → process → predict → select → append → repeat. One token at a time builds the full response.

APPLIED AI FOR DEVELOPERS

Key Takeaways & What's Next

You've learned how LLMs turn text into tokens, process them, and generate text one token at a time.

Remember: Next-token predictions depend on learned patterns **and the current context.**

Key Takeaways

1. **Text becomes tokens**
   Your input is split into small pieces called tokens.
2. **Tokens get IDs**
   Each token maps to an ID in the vocabulary.
3. **IDs become vectors**
   Token IDs map to learned embeddings the model can process.
4. **The model processes**
   A Transformer uses attention and layers to process context.
5. **It predicts probabilities**
   The model outputs a distribution over the entire vocabulary.
6. **It selects the next token**
   A decoding strategy chooses one token as the next output.
7. **It appends and repeats**
   Add the token and repeat until a stopping condition is met.
8. **You get the final output**
   The sequence of selected tokens forms the generated response.

What's Next in This Tutorial Series

II LLM Fundamentals II
Attention, positional encoding, context windows, training vs. inference, and model limitations.
Focus: Architecture & Internals

III Embeddings & Semantic Search
Text embeddings, vector similarity, semantic search and RAG.
Focus: Meaning & Retrieval

IV Building with LLMs
Prompt engineering, tool use, function calling and reliable LLM applications.
Focus: Practical Skills

V Production & Best Practices
Evaluation, safety, observability, cost optimization and deployment at scale.
Focus: Reliability & Scale

Keep Building:
The more you build with LLMs, the better you'll understand their strengths and limitations. Experiment, iterate, and keep learning!