Tokens and Context Windows

What tokens and tokenization are, the hard limit of the context window, what happens when context fills up, and truncation/summarization/retrieval strategies for managing limited context.

Intermediate 18 min

Last updated: August 20, 2026
Read online: https://www.learnforgex.com/en/topics/tokens-and-context-windows

---

Tokens and Context Windows

"How Large Language Models Work" introduced two ideas this lesson goes much deeper on: **tokens** (the small units a model predicts one at a time) and **context** (everything the model sees before generating). Both turn out to have hard, practical limits that shape almost everything about working with an LLM -- how much information you can give it at once, how much a request costs, and why a model can "forget" something from earlier in a very long conversation. Understanding these limits is essential groundwork for the "Prompting and Prompt Engineering" lesson right after this one, and for the "Tools & MCP" category later in this course, which exists largely to manage what gets into a model's limited context.

What Is a Token, Exactly?

"How Large Language Models Work" described a token as "a smaller unit" than a word, without saying exactly what that means -- this lesson makes it precise. A token is the actual unit of text an LLM reads and generates; it is often, but not always, a whole word. Common words ("the," "cat") are frequently single tokens; less common or longer words often get split into multiple tokens (for example, "tokenization" might become "token" + "ization"); punctuation, spaces, and even parts of emoji or non-English text are also tokens. This splitting process is called **tokenization**, and it happens via a fixed vocabulary the model was trained with -- a rough rule of thumb for English text is that a token is about 4 characters, or that 100 tokens is roughly 75 words, though this varies noticeably by language and content type (code and non-English text often use more tokens per word than English prose).

> Because tokenization varies by language, the same sentence can cost noticeably more tokens in one language than another -- a real, practical consideration for cost and context budget if you're building something in more than one language, not just an academic detail.

Why Tokens Instead of Whole Words?

Recall from "How Large Language Models Work" that pretraining trains a model to predict the next token given everything before it -- the model needs a fixed, finite vocabulary of possible tokens to predict from. A vocabulary of whole words alone would either be enormous (to cover every word in every language, including names, typos, and made-up words) or would constantly fail on words it had never seen. Splitting text into smaller, reusable pieces solves both problems: a modest vocabulary (commonly tens of thousands of entries) of common words, word-pieces, and individual characters can represent literally any text, including words the tokenizer's designers never anticipated, by falling back to smaller pieces.

The Context Window: A Hard Limit

"A First Look at Context" in the previous lesson described context as everything a model sees before generating its next token. That "everything" has a hard ceiling called the **context window** -- the maximum number of tokens a specific model can process at once, combining your instructions, any supplied background information, the conversation history, and the response being generated. Context windows vary by model, ranging from a few thousand tokens in early LLMs to hundreds of thousands (or more) in modern ones -- but every model has *some* limit. Once a conversation's total token count would exceed that limit, something has to give: older content gets dropped, summarized, or the request is rejected outright, depending on how the system calling the model is built.

What Happens When Context Fills Up

This is where the "model forgets earlier parts of a long conversation" experience most people have run into actually comes from. Recall from "In-Context Learning: How a Model Uses What You Give It" that a model has no memory of its own -- everything it "knows" about the current conversation is whatever text is present in its context right now. If the system feeding the model has to drop the earliest messages to stay under the context window limit, that information is not "forgotten" by the model in any deeper sense -- it's simply no longer part of the text the model is reading, so it has no way to use it, exactly the same as if it had never been said. This isn't a flaw specific to any one product; it's a direct, unavoidable consequence of the context window being finite.

Tokens as a Cost and Latency Unit

Beyond the context window limit, tokens are also the practical unit that LLM usage is measured and often billed by -- both the tokens you send (input) and the tokens the model generates (output). This has two very concrete implications for anyone building on top of an LLM: longer prompts and longer conversation histories cost more per request, and generating a longer response takes measurably more time, because (recall from "In-Context Learning: How a Model Uses What You Give It") the model re-reads the entire context on every single response. Being deliberate about what actually needs to be in context -- rather than including everything "just in case" -- is therefore not just a correctness concern but a real cost and speed concern, a theme the "Tools & MCP" category returns to when it covers how systems select what information to give a model.

Managing a Limited Context Window

Since every model has a hard token ceiling, real systems built on LLMs use several recurring strategies to work within it, each with a real trade-off:

- **Truncation:** simply dropping the oldest messages once the limit is approached -- simple, but risks losing genuinely important earlier context (see "What Happens When Context Fills Up").
- **Summarization:** periodically replacing older conversation history with a shorter summary of it -- preserves the gist while using far fewer tokens, at the cost of losing exact wording and detail.
- **Retrieval:** rather than keeping everything in context all the time, fetching only the specific pieces of information relevant to the current request and inserting just those into context -- this is the foundational idea behind techniques this course's later categories build on extensively.

No single strategy is universally correct -- which one (or combination) makes sense depends entirely on what the specific application actually needs to remember.

Best Practices

- Don't treat context as free or unlimited -- every token you include (see "Tokens as a Cost and Latency Unit") has a real cost in both money and response time, on top of counting against the hard context window ceiling.
- When a conversation needs to span more information than comfortably fits in a context window, decide deliberately between truncation, summarization, or retrieval (see "Managing a Limited Context Window") rather than letting a system silently drop whatever happens to be oldest.

- Remember that token count is not the same as word or character count (see "What Is a Token, Exactly?") -- when estimating how much fits in a context window, count tokens, not words.

Common Mistakes

- **Assuming a model "remembers" something it was told much earlier in a very long conversation.** As "What Happens When Context Fills Up" explained, once content falls out of the context window, the model has no access to it at all -- there's no deeper memory to fall back on.
- **Assuming all languages or content types cost roughly the same number of tokens for the same text.** As the tip in "What Is a Token, Exactly?" noted, tokenization varies meaningfully by language and content type.
- **Treating a larger context window as removing the need to think about what's included.** Even with a very large context window, including irrelevant information still costs tokens (money and latency, see "Tokens as a Cost and Latency Unit") and can distract a model from what actually matters in a request.

Summary, Cheat Sheet, and Glossary

Summary

- A **token** is the actual unit of text an LLM reads and generates -- often close to a word, but not always -- produced by a process called **tokenization**.
- The **context window** is the hard maximum number of tokens a model can process at once, combining instructions, background information, conversation history, and the response itself.
- When context fills up, older content must be dropped, summarized, or otherwise handled -- the model has no memory beyond what's actually in its context right now.
- Tokens are also the practical unit LLM usage is measured and billed by, making context size a real cost and latency concern, not just a correctness one.
- Truncation, summarization, and retrieval are the three common strategies for working within a limited context window, each with different trade-offs.

Cheat Sheet

- Token ≈ often close to a word, but can be smaller (word-piece, punctuation, character).
- Context window = the hard token ceiling for a given model.
- Context full → truncate (drop oldest), summarize (compress), or retrieve (fetch only what's relevant).
- More tokens in context = more cost + more latency, always.

Glossary

- **Token:** the actual unit of text an LLM reads and generates, produced by tokenization -- often close to a word, but sometimes smaller.
- **Tokenization:** the process of splitting text into tokens using a model's fixed vocabulary.
- **Context window:** the maximum number of tokens a specific model can process in a single request, combining all context sources.
- **Truncation:** dropping the oldest content from context once the token limit is approached.
- **Summarization (in this context):** replacing older conversation history with a shorter summary to save tokens.
- **Retrieval:** fetching only the specific, relevant pieces of information needed for the current request instead of keeping everything in context.