READ MODE · ANSWER FIRST

How real-time LLM token streaming works

See how an LLM response travels from token generation through SSE, HTTP, the network stack, and flow control to reach your screen.

CHEAT SHEET · 5 ESSENTIAL IDEAS

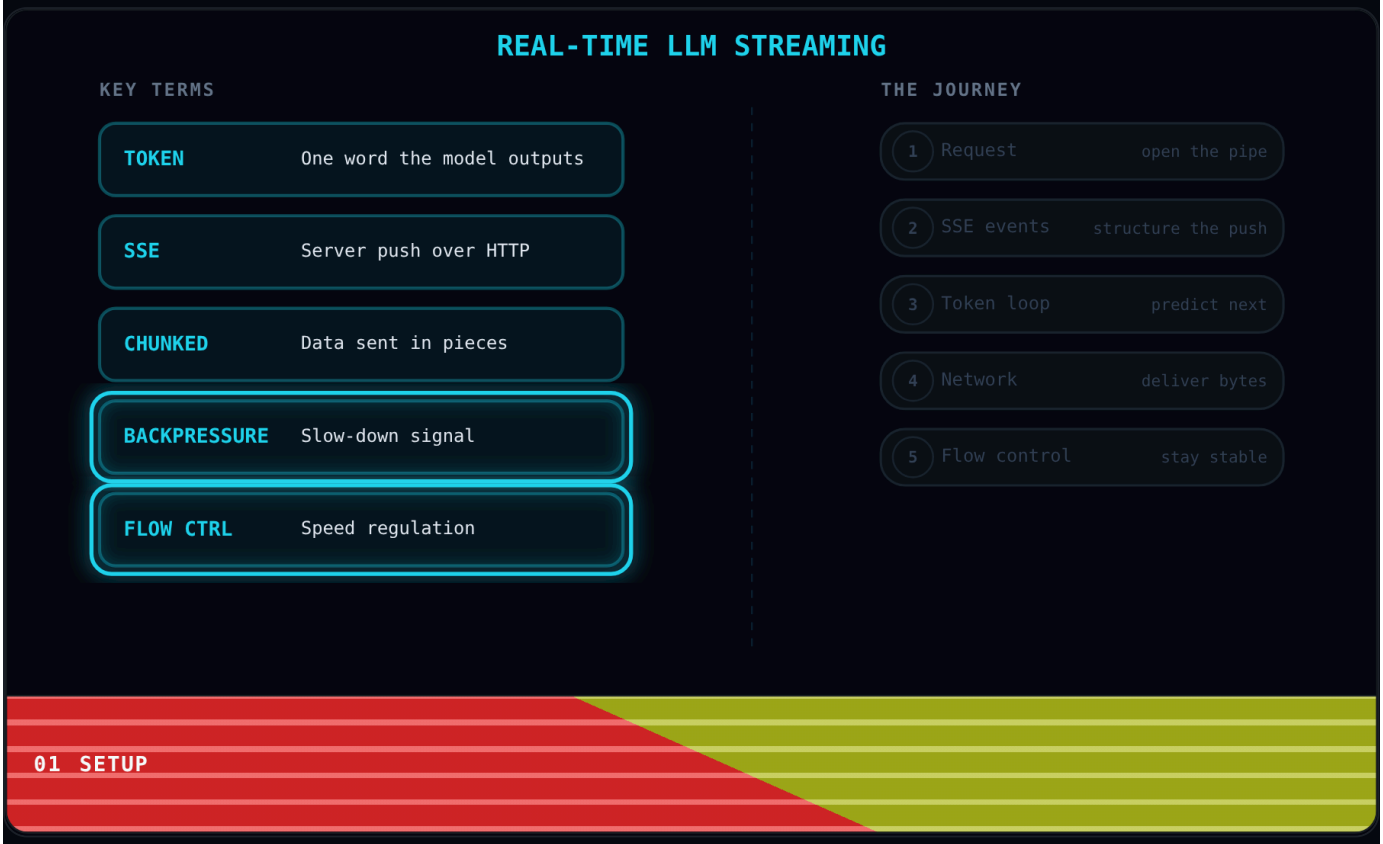
The whole story in 5 lines

LLM streaming is a self-regulating pipeline that generates, packages, transports, and displays one token at a time.

1. The client asks for streaming and the server answers with headers that keep the response open.
2. Each event starts with data, ends with a blank line, and carries one token payload.
3. Every output becomes part of the next input, so response generation is inherently sequential.
4. Each layer adds a different guarantee: framing, privacy, or ordered delivery.
5. Backpressure travels upstream one buffer at a time until generation matches consumption.

STAGE 1 OF 7

Setup



When an AI reply appears word by word, you are watching a real streaming pipeline. We will follow one response from the first request to the final safety checks.

A token is a small piece of text produced by the model. Streaming sends each token to the screen as soon as it exists instead of waiting for the full answer.

Server-Sent Events, or SSE, let a server push messages over one HTTP response. Think of it as a one-way radio: the server talks while the client listens.

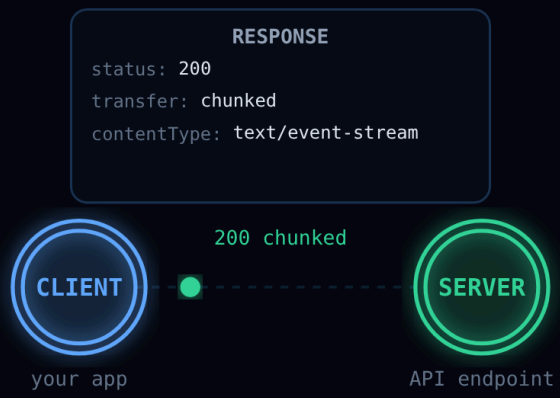
Chunked delivery sends a response in pieces. The browser can read an early chunk while the server is still preparing later chunks.

Backpressure signals that a receiver is falling behind. Flow control uses that signal to slow the sender before queues overflow.

Our journey follows the request, SSE events, token generation, network delivery, and backpressure. Let us begin by opening the streaming connection.

STAGE 2 OF 7

The Streaming Request



02 THE STREAMING REQUEST

The client has a prompt ready. The way it asks for the answer determines whether the user waits in silence or sees tokens immediately.

The client sends an HTTP POST containing the prompt, model name, and stream set to true. That flag asks the server to return pieces as they become ready.

The server accepts the request and returns a successful response whose content type is text/event-stream. The client now knows it should keep listening.

Chunked delivery lets the first response pieces travel before the server knows the final size. The open connection can now carry tokens as they are produced.

Switch the Mode control to Batch and the client waits for a complete response. Switch it to Streaming and the first piece can arrive while generation continues.

The pipe is open. Next we need a simple format that separates one token message from the next, which is the job of SSE.

STAGE 3 OF 7

Server-Sent Events

The open response now needs boundaries between token messages. SSE supplies a tiny text format that both servers and browsers understand.

An SSE event begins with data followed by a payload and ends with a blank line. That blank line tells the client where one event stops.

Adjust the Tokens control and the visible event stream grows or shrinks. Every event keeps its own index and token text.

After the last token, the server sends data: [DONE]. The client treats that marker as the end of the streamed response.

SSE stays simple because it uses ordinary HTTP and only sends messages from server to client. That is exactly the direction token streaming needs.

SSE provides the envelopes. Next we will look inside the server to see how the model creates the tokens placed in them.

STAGE 4 OF 7

Token-by-Token Generation

SSE can deliver tokens only after the model creates them. Inside the server, an autoregressive loop predicts one new token at a time.

The loop starts with the prompt. The model reads that context and predicts the first response token.

The first token moves into the growing buffer and a copy can enter the SSE stream. What gives the model enough context to choose the next token?

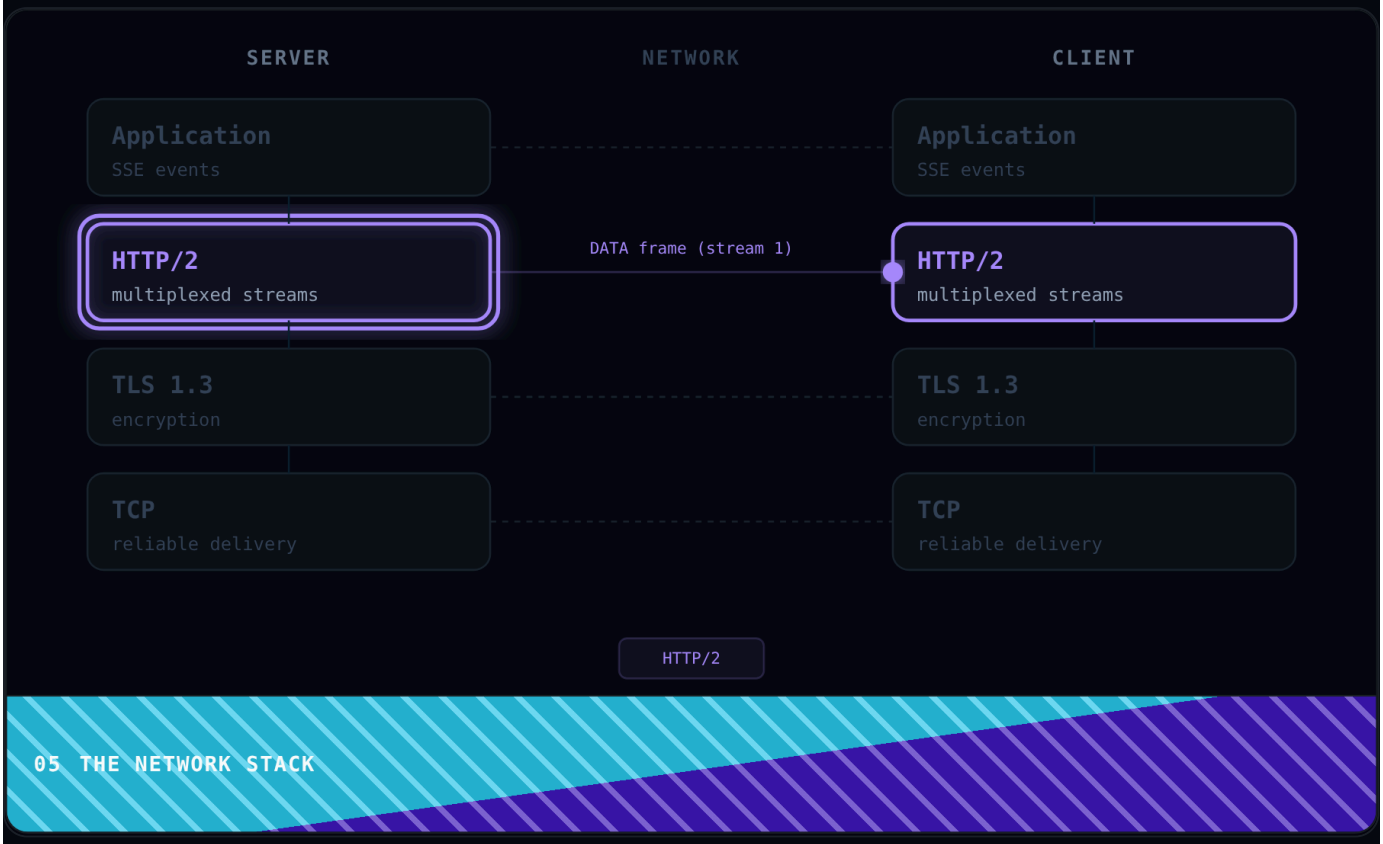
The generated token returns as part of the next input. The model now predicts again from the prompt plus everything produced so far.

Switch the Model speed control to compare illustrative rates of about 80 and 15 tokens per second. The loop stays sequential in both cases.

Each new token enters the SSE encoder and then the open HTTP response. Next we will trace the bytes through the network stack.

STAGE 5 OF 7

The Network Stack



05 THE NETWORK STACK

A token has entered its SSE envelope. Now we will follow those bytes through the protocol layers between the server and browser.

The application layer holds the SSE text. HTTP turns that payload into frames or chunks that can travel over a connection.

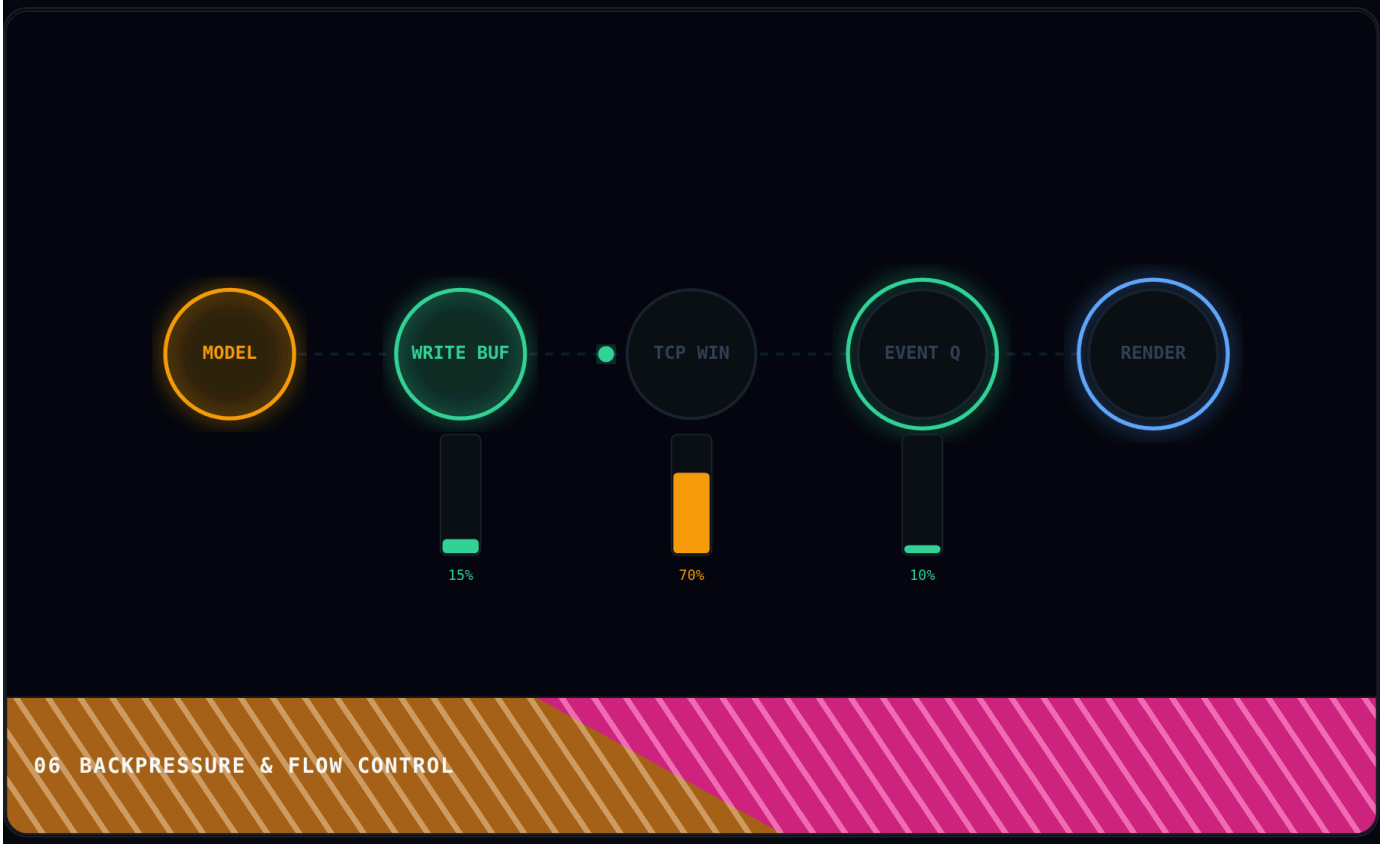
Switch the HTTP version control. HTTP/2 shares one connection across streams while HTTP/1.1 uses separate connections in this comparison.

TLS encrypts the HTTP bytes before they leave the server. Observers between the endpoints cannot read the token text.

TCP keeps bytes ordered and retransmits missing data. That reliability can briefly stall delivery when a packet is lost.

The browser unwraps the layers and reconstructs SSE events. Next we will see how the pipeline reacts when the client cannot keep up.

Backpressure & Flow Control



06 BACKPRESSURE & FLOW CONTROL

The network delivers tokens reliably, but the client may process them more slowly than the server produces them. Buffers absorb that difference for a while.

When producer and consumer rates match, the server buffer, TCP windows, and browser queue remain mostly empty.

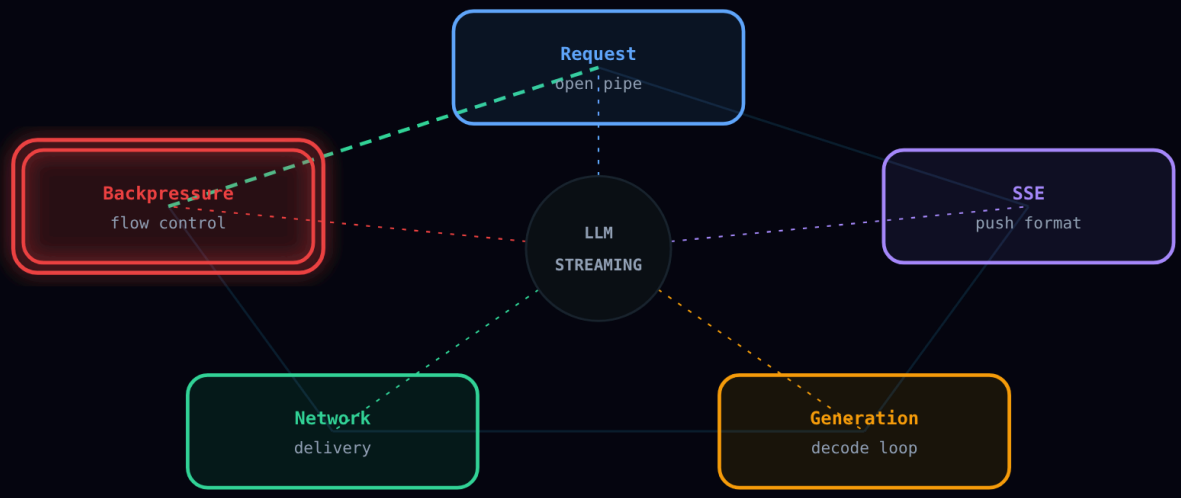
Switch the Client speed control to Slow and the queues fill while receive windows shrink. Switch it to Fast and the queued work drains.

A shrinking receive window prevents the server from sending more bytes. Its own write buffer then starts to fill.

HTTP/2 also gives each stream a byte window. WINDOW_UPDATE messages grant more space without blocking unrelated streams.

Each layer watches its own capacity and slows the layer before it. That chain keeps the stream stable. Now let us connect the whole journey.

Recap



07 RECAP

We followed one answer through the complete streaming pipeline. Let us reconnect the five mechanisms that made immediate delivery possible.

We started with a request whose stream flag opened a response before the full answer existed.

SSE then wrapped each generated token in a small text event and marked the end with a DONE signal.

Inside the server, every generated token became part of the next prediction. That return path made the decode loop sequential.

HTTP framed the events, TLS protected them, and TCP delivered their bytes in order to the browser.

Backpressure connected the whole system in reverse. A slow client filled buffers and eventually forced the producer to slow down.

Real-time LLM streaming is a chain of small handoffs: generate, package, transport, display, and regulate. Together they make the answer feel immediate.