`streamText()`

Streams text generations from a language model.

You can use the streamText function for interactive use cases such as chat bots and other real-time applications. You can also generate UI components with tools.

```
import { openai } from '@ai-sdk/openai';
import { streamText } from 'ai';

const { textStream } = streamText({
  model: openai('gpt-4-turbo'),
  prompt: 'Invent a new holiday and describe its traditions.',
});

for await (const textPart of textStream) {
  process.stdout.write(textPart);
}
```

To see `streamText` in action, check out these examples.

Import

```
import { streamText } from "ai"
```

API Signature

Parameters

`model: LanguageModel`

The language model to use. Example: openai('gpt-4-turbo')

`system: string`

The system prompt to use that specifies the behavior of the model.

`prompt: string`

The input prompt to generate the text from.

`messages: Array<CoreSystemMessage | CoreUserMessage | CoreAssistantMessage | CoreToolMessage> | Array<UIMessage>`

A list of messages that represent a conversation. Automatically converts UI messages from the useChat hook.

CoreSystemMessage

`role: 'system'`

The role for the system message.

`content: string`

The content of the message.

CoreUserMessage

`role: 'user'`

The role for the user message.

`content: string | Array<TextPart | ImagePart | FilePart>`

The content of the message.

TextPart

`type: 'text'`

The type of the message part.

`text: string`

The text content of the message part.

ImagePart

`type: 'image'`

The type of the message part.

`image: string | Uint8Array | Buffer | ArrayBuffer | URL`

The image content of the message part. String are either base64 encoded content, base64 data URLs, or http(s) URLs.

`mimeType?: string`

The mime type of the image. Optional.

`FilePart`

`type: 'file'`

The type of the message part.

`data: string | Uint8Array | Buffer | ArrayBuffer | URL`

The file content of the message part. String are either base64 encoded content, base64 data URLs, or http(s) URLs.

`mimeType: string`

The mime type of the file.

`CoreAssistantMessage`

`role: 'assistant'`

The role for the assistant message.

`content: string | Array<TextPart | ToolCallPart>`

The content of the message.

`TextPart`

`type: 'text'`

The type of the message part.

`text: string`

The text content of the message part.

`ToolCallPart`

`type: 'tool-call'`

The type of the message part.

`toolCallId: string`

The id of the tool call.

`toolName: string`

The name of the tool, which typically would be the name of the function.

`args: object based on zod schema`

Parameters generated by the model to be used by the tool.

CoreToolMessage

`role: 'tool'`
The role for the assistant message.

`content: Array<ToolResultPart>`
The content of the message.

ToolResultPart

`type: 'tool-result'`
The type of the message part.

`toolCallId: string`
The id of the tool call the result corresponds to.

`toolName: string`
The name of the tool the result corresponds to.

`result: unknown`
The result returned by the tool after execution.

`isError?: boolean`
Whether the result is an error or an error message.

`tools: Record<string, CoreTool>`
Tools that are accessible to and can be called by the model. The model needs to support calling tools.

CoreTool

`description?: string`
Information about the purpose of the tool including details on how and when it can be used by the model.

`parameters: Zod Schema | JSON Schema`
The schema of the input that the tool expects. The language model will use this to generate the input. It is also used to validate the output of the language model. Use descriptions to make the input understandable for the language model. You can either pass in a Zod schema or a JSON schema (using the `jsonSchema` function).

`execute?: async (parameters: T, options: ToolExecutionOptions) ⇒ RESULT`
An async function that is called with the arguments from the tool call and produces a result. If not provided, the tool will not be executed automatically.

ToolExecutionOptions

`toolCallId: string`

The ID of the tool call. You can use it e.g. when sending tool-call related information with stream data.

`messages: CoreMessage[]`

Messages that were sent to the language model to initiate the response that contained the tool call. The messages do not include the system prompt nor the assistant response that contained the tool call.

`abortSignal: AbortSignal`

An optional abort signal that indicates that the overall operation should be aborted.

`toolChoice?: "auto" | "none" | "required" | { "type": "tool", "toolName": string }`

The tool choice setting. It specifies how tools are selected for execution. The default is "auto". "none" disables tool execution. "required" requires tools to be executed. { "type": "tool", "toolName": string } specifies a specific tool to execute.

`maxTokens?: number`

Maximum number of tokens to generate.

`temperature?: number`

Temperature setting. The value is passed through to the provider. The range depends on the provider and model. It is recommended to set either `temperature` or `topP`, but not both.

`topP?: number`

Nucleus sampling. The value is passed through to the provider. The range depends on the provider and model. It is recommended to set either `temperature` or `topP`, but not both.

`topK?: number`

Only sample from the top K options for each subsequent token. Used to remove "long tail" low probability responses. Recommended for advanced use cases only. You usually only need to use temperature.

`presencePenalty?: number`

Presence penalty setting. It affects the likelihood of the model to repeat information that is already in the prompt. The value is passed through to the provider. The range depends on the provider and model.

`frequencyPenalty?: number`

Frequency penalty setting. It affects the likelihood of the model to repeatedly use the same words or phrases. The value is passed through to the provider. The range depends on the provider and model.

`stopSequences?: string[]`

Sequences that will stop the generation of the text. If the model generates any of these sequences, it will stop generating further text.

`seed?: number`

The seed (integer) to use for random sampling. If set and supported by the model, calls will generate deterministic results.

`maxRetries?: number`

Maximum number of retries. Set to 0 to disable retries. Default: 2.

`abortSignal?: AbortSignal`

An optional abort signal that can be used to cancel the call.

`headers?: Record<string, string>`

Additional HTTP headers to be sent with the request. Only applicable for HTTP-based providers.

`maxSteps?: number`

Maximum number of sequential LLM calls (steps), e.g. when you use tool calls. A maximum number is required to prevent infinite loops in the case of misconfigured tools. By default, it is set to 1.

`experimental_continueSteps?: boolean`

Enable or disable continue steps. Disabled by default.

`experimental_telemetry?: TelemetrySettings`

Telemetry configuration. Experimental feature.

`TelemetrySettings`

`isEnabled?: boolean`

Enable or disable telemetry. Disabled by default while experimental.

`recordInputs?: boolean`

Enable or disable input recording. Enabled by default.

`recordOutputs?: boolean`

Enable or disable output recording. Enabled by default.

`functionId?: string`

Identifier for this function. Used to group telemetry data by function.

```
metadata?: Record<string, string | number | boolean | Array<null |
           undefined | string> | Array<null | undefined | number> |
           Array<null | undefined | boolean>>
```

Additional information to include in the telemetry data.

`experimental_toolCallStreaming?: boolean`

Enable streaming of tool call deltas as they are generated. Disabled by default.

`experimental_providerMetadata?: Record<string,Record<string,JSONValue>> | undefined`

Optional metadata from the provider. The outer key is the provider name. The inner values are the metadata. Details depend on the provider.

`experimental_activeTools?: Array<TOOLNAME> | undefined`

The tools that are currently active. All tools are active by default.

`experimental_repairToolCall?: (options: ToolCallRepairOptions) => Promise<LanguageModelV1FunctionToolCall | null>`

A function that attempts to repair a tool call that failed to parse. Return either a repaired tool call or null if the tool call cannot be repaired.

ToolCallRepairOptions

`system: string | undefined`

The system prompt.

`messages: CoreMessage[]`

The messages in the current generation step.

`toolCall: LanguageModelV1FunctionToolCall`

The tool call that failed to parse.

`tools: TOOLS`

The tools that are available.

`parameterSchema: (options: { toolName: string }) => JSONSchema7`

A function that returns the JSON Schema for a tool.

`error: NoSuchToolError | InvalidToolArgumentsError`

The error that occurred while parsing the tool call.

`onChunk?: (event: OnChunkResult) => Promise<void> |void`

Callback that is called for each chunk of the stream. The stream processing will pause until the callback promise is resolved.

OnChunkResult

`chunk: TextStreamPart`

The chunk of the stream.

TextStreamPart

`type: 'text-delta'`

The type to identify the object as text delta.

`textDelta: string`

The text delta.

TextStreamPart

`type: 'tool-call'`

The type to identify the object as tool call.

`toolCallId: string`

The id of the tool call.

`toolName: string`

The name of the tool, which typically would be the name of the function.

`args: object based on zod schema`

Parameters generated by the model to be used by the tool.

TextStreamPart

`type: 'tool-call-streaming-start'`

Indicates the start of a tool call streaming. Only available when streaming tool calls.

`toolCallId: string`

The id of the tool call.

`toolName: string`

The name of the tool, which typically would be the name of the function.

TextStreamPart

`type: 'tool-call-delta'`

The type to identify the object as tool call delta. Only available when streaming tool calls.

`toolCallId: string`

The id of the tool call.

`toolName: string`

The name of the tool, which typically would be the name of the function.

`argsTextDelta: string`

The text delta of the tool call arguments.

TextStreamPart

`type: 'tool-result'`

The type to identify the object as tool result.

`toolCallId: string`

The id of the tool call.

`toolName: string`

The name of the tool, which typically would be the name of the function.

`args: object based on zod schema`

Parameters generated by the model to be used by the tool.

`result: any`

The result returned by the tool after execution has completed.

`onStepFinish?: (result: onStepFinishResult) ⇒ Promise<void> | void`

Callback that is called when a step is finished.

onStepFinishResult

`stepType: "initial" | "continue" | "tool-result"`

The type of step. The first step is always an "initial" step, and subsequent steps are either "continue" steps or "tool-result" steps.

```
finishReason: "stop" | "length" | "content-filter" | "tool-calls" |
              "error" | "other" | "unknown"
```

The reason the model finished generating the text for the step.

`usage: TokenUsage`

The token usage of the step.

TokenUsage

`promptTokens: number`

The total number of tokens in the prompt.

`completionTokens: number`

The total number of tokens in the completion.

`totalTokens: number`

The total number of tokens generated.

`text: string`

The full text that has been generated.

`toolCalls: ToolCall[]`

The tool calls that have been executed.

`toolResults: ToolResult[]`

The tool results that have been generated.

`warnings: Warning[] | undefined`

Warnings from the model provider (e.g. unsupported settings).

`response?: Response`

Response metadata.

Response

`id: string`

The response identifier. The AI SDK uses the ID from the provider response when available, and generates an ID otherwise.

`model: string`

The model that was used to generate the response. The AI SDK uses the response model from the provider response when available, and the model from the function call otherwise.

`timestamp: Date`

The timestamp of the response. The AI SDK uses the response timestamp from the provider response when available, and creates a timestamp otherwise.

`headers?: Record<string, string>`

Optional response headers.

`isContinued: boolean`

True when there will be a continuation step with a continuation text.

`experimental_providerMetadata?: Record<string,Record<string,JSONValue>> | undefined`

Optional metadata from the provider. The outer key is the provider name. The inner values are the metadata. Details depend on the provider.

`onFinish?: (result: OnFinishResult) => Promise<void> | void`

Callback that is called when the LLM response and all request tool executions (for tools that have an `execute` function) are finished.

OnFinishResult

`finishReason: "stop" | "length" | "content-filter" | "tool-calls" | "error" | "other" | "unknown"`

The reason the model finished generating the text.

`usage: TokenUsage`

The token usage of the generated text.

TokenUsage

`promptTokens: number`

The total number of tokens in the prompt.

`completionTokens: number`

The total number of tokens in the completion.

`totalTokens: number`

The total number of tokens generated.

`experimental_providerMetadata: Record<string,Record<string,JSONValue>> | undefined`

Optional metadata from the provider. The outer key is the provider name. The inner values are the metadata. Details depend on the provider.

`text: string`

The full text that has been generated.

`toolCalls: ToolCall[]`

The tool calls that have been executed.

`toolResults: ToolResult[]`

The tool results that have been generated.

`warnings: Warning[] | undefined`

Warnings from the model provider (e.g. unsupported settings).

`response?: Response`

Response metadata.

Response

`id: string`

The response identifier. The AI SDK uses the ID from the provider response when available, and generates an ID otherwise.

`model: string`

The model that was used to generate the response. The AI SDK uses the response model from the provider response when available, and the model from the function call otherwise.

`timestamp: Date`

The timestamp of the response. The AI SDK uses the response timestamp from the provider response when available, and creates a timestamp otherwise.

`headers?: Record<string, string>`

Optional response headers.

`messages: Array<CoreAssistantMessage | CoreToolMessage>`

The response messages that were generated during the call. It consists of an assistant message, potentially containing tool calls. When there are tool results, there is an additional tool message with the tool results that are available. If there are tools that do not have execute functions, they are not included in the tool results and need to be added separately.

`steps: Array<StepResult>`

Response information for every step. You can use this to get information about intermediate steps, such as the tool calls or the response headers.

Returns

`finishReason: Promise<'stop' | 'length' | 'content-filter' | 'tool-calls' | 'error' | 'other' | 'unknown'>`

The reason why the generation finished. Resolved when the response is finished.

`usage: Promise<CompletionTokenUsage>`

The token usage of the generated text. Resolved when the response is finished.

CompletionTokenUsage

`promptTokens: number`

The total number of tokens in the prompt.

`completionTokens: number`

The total number of tokens in the completion.

`totalTokens: number`

The total number of tokens generated.

`experimental_providerMetadata: Promise<Record<string,Record<string,JSONValue>> | undefined>`

Optional metadata from the provider. Resolved whe the response is finished. The outer key is the provider name. The inner values are the metadata. Details depend on the provider.

`responseMessages: Promise<Array<CoreAssistantMessage | CoreToolMessage>>`

The response messages that were generated during the call. It consists of an assistant message, potentially containing tool calls. When there are tool results, there is an additional tool message with the tool results that are available. If there are tools that do not have execute functions, they are not included in the tool results and need to be added separately. Resolved when the response is finished.

`text: Promise<string>`

The full text that has been generated. Resolved when the response is finished.

`toolCalls: Promise<ToolCall[]>`

The tool calls that have been executed. Resolved when the response is finished.

`toolResults: Promise<ToolResult[]>`

The tool results that have been generated. Resolved when the all tool executions are finished.

`request?: Promise<RequestMetadata>`

Request metadata.

`RequestMetadata`

`body: string`

Raw request HTTP body that was sent to the provider API as a string (JSON should be stringified).

`response?: Promise<ResponseMetadata>`

Response metadata. Resolved when the response is finished.

`ResponseMetadata`

`id: string`

The response identifier. The AI SDK uses the ID from the provider response when available, and generates an ID otherwise.

`model: string`

The model that was used to generate the response. The AI SDK uses the response model from the provider response when available, and the model from the function call otherwise.

`timestamp: Date`

The timestamp of the response. The AI SDK uses the response timestamp from the provider response when available, and creates a timestamp otherwise.

`headers?: Record<string, string>`

Optional response headers.

`messages: Array<CoreAssistantMessage | CoreToolMessage>`

The response messages that were generated during the call. It consists of an assistant message, potentially containing tool calls. When there are tool results, there is an additional tool message with the tool results that are available. If there are tools that do not have execute functions, they are not included in the tool results and need to be added separately.

`warnings: Warning[] | undefined`

Warnings from the model provider (e.g. unsupported settings).

`steps: Promise<Array<StepResult>>`

Response information for every step. You can use this to get information about intermediate steps, such as the tool calls or the response headers.

StepResult

`stepType: "initial" | "continue" | "tool-result"`

The type of step. The first step is always an "initial" step, and subsequent steps are either "continue" steps or "tool-result" steps.

`text: string`

The generated text by the model.

`toolCalls: array`

A list of tool calls made by the model.

`toolResults: array`

A list of tool results returned as responses to earlier tool calls.

```
finishReason: 'stop' | 'length' | 'content-filter' | 'tool-calls' |
              'error' | 'other' | 'unknown'
```

The reason the model finished generating the text.

`usage: CompletionTokenUsage`

The token usage of the generated text.

CompletionTokenUsage

`promptTokens: number`

The total number of tokens in the prompt.

`completionTokens: number`

The total number of tokens in the completion.

`totalTokens: number`

The total number of tokens generated.

`request?: RequestMetadata`

Request metadata.

RequestMetadata

`body: string`

Raw request HTTP body that was sent to the provider API as a string (JSON should be stringified).

`response?: ResponseMetadata`

Response metadata.

ResponseMetadata

`id: string`

The response identifier. The AI SDK uses the ID from the provider response when available, and generates an ID otherwise.

`model: string`

The model that was used to generate the response. The AI SDK uses the response model from the provider response when available, and the model from the function call otherwise.

`timestamp: Date`

The timestamp of the response. The AI SDK uses the response timestamp from the provider response when available, and creates a timestamp otherwise.

`headers?: Record<string, string>`

Optional response headers.

`messages: Array<CoreAssistantMessage | CoreToolMessage>`

The response messages that were generated during the call. It consists of an assistant message, potentially containing tool calls. When there are tool results, there is an additional tool message with the tool results that are available. If there are tools that do not have execute functions, they are not included in the tool results and need to be added separately.

`warnings: Warning[] | undefined`

Warnings from the model provider (e.g. unsupported settings).

`isContinued: boolean`

True when there will be a continuation step with a continuation text.

`experimental_providerMetadata?: Record<string,Record<string,JSONValue>> | undefined`

Optional metadata from the provider. The outer key is the provider name. The inner values are the metadata. Details depend on the provider.

`textStream: AsyncIterable<string> & ReadableStream<string>`

A text stream that returns only the generated text deltas. You can use it as either an AsyncIterable or a ReadableStream. When an error occurs, the stream will throw the error.

`fullStream: AsyncIterable<TextStreamPart> & ReadableStream<TextStreamPart>`

A stream with all events, including text deltas, tool calls, tool results, and errors. You can use it as either an AsyncIterable or a ReadableStream. Only errors that stop the stream, such as network errors, are thrown.

TextStreamPart

`type: 'text-delta'`

The type to identify the object as text delta.

`textDelta: string`
The text delta.

TextStreamPart

`type: 'tool-call'`
The type to identify the object as tool call.

`toolCallId: string`
The id of the tool call.

`toolName: string`
The name of the tool, which typically would be the name of the function.

`args: object based on zod schema`
Parameters generated by the model to be used by the tool.

TextStreamPart

`type: 'tool-call-streaming-start'`
Indicates the start of a tool call streaming. Only available when streaming tool calls.

`toolCallId: string`
The id of the tool call.

`toolName: string`
The name of the tool, which typically would be the name of the function.

TextStreamPart

`type: 'tool-call-delta'`
The type to identify the object as tool call delta. Only available when streaming tool calls.

`toolCallId: string`
The id of the tool call.

`toolName: string`
The name of the tool, which typically would be the name of the function.

`argsTextDelta: string`
The text delta of the tool call arguments.

TextStreamPart

`type: 'tool-result'`

The type to identify the object as tool result.

`toolCallId: string`

The id of the tool call.

`toolName: string`

The name of the tool, which typically would be the name of the function.

`args: object based on zod schema`

Parameters generated by the model to be used by the tool.

`result: any`

The result returned by the tool after execution has completed.

TextStreamPart

`type: 'error'`

The type to identify the object as error.

`error: Error`

Describes the error that may have occurred during execution.

TextStreamPart

`type: 'step-finish'`

The type to identify the object as step finish.

```
finishReason: 'stop' | 'length' | 'content-filter' | 'tool-calls' |
              'error' | 'other' | 'unknown'
```

The reason the model finished generating the text.

`usage: TokenUsage`

The token usage of the generated text.

TokenUsage

`promptTokens: number`

The total number of tokens in the prompt.

`completionTokens: number`

The total number of tokens in the completion.

`totalTokens: number`

The total number of tokens generated.

`isContinued: boolean`

True when there will be a continuation step with a continuation text.

`response?: Response`

Response metadata.

Response

`id: string`

The response identifier. The AI SDK uses the ID from the provider response when available, and generates an ID otherwise.

`model: string`

The model that was used to generate the response. The AI SDK uses the response model from the provider response when available, and the model from the function call otherwise.

`timestamp: Date`

The timestamp of the response. The AI SDK uses the response timestamp from the provider response when available, and creates a timestamp otherwise.

TextStreamPart

`type: 'finish'`

The type to identify the object as finish.

```
finishReason: 'stop' | 'length' | 'content-filter' | 'tool-calls' |
              'error' | 'other' | 'unknown'
```

The reason the model finished generating the text.

`usage: TokenUsage`

The token usage of the generated text.

TokenUsage

`promptTokens: number`

The total number of tokens in the prompt.

`completionTokens: number`

The total number of tokens in the completion.

`totalTokens: number`

The total number of tokens generated.

`response?: Response`

Response metadata.

Response

`id: string`

The response identifier. The AI SDK uses the ID from the provider response when available, and generates an ID otherwise.

`model: string`

The model that was used to generate the response. The AI SDK uses the response model from the provider response when available, and the model from the function call otherwise.

`timestamp: Date`

The timestamp of the response. The AI SDK uses the response timestamp from the provider response when available, and creates a timestamp otherwise.

`pipeDataStreamToResponse: (response: ServerResponse, options: PipeDataStreamToResponseOptions } ⇒ void`

Writes stream data output to a Node.js response-like object. It sets a `Content-Type` header to `text/plain; charset=utf-8` and writes each stream data part as a separate chunk.

`PipeDataStreamToResponseOptions`

`status?: number`

The response status code.

`statusText?: string`

The response status text.

`headers?: HeadersInit`

The response headers.

`data?: StreamData`

The stream data object.

`getErrorMessage?: (error: unknown) ⇒ string`

A function to get the error message from the error object. By default, all errors are masked as "" for safety reasons.

`sendUsage?: boolean`

Whether to send the usage information in the stream. Defaults to true.

`pipeTextStreamToResponse: (response: ServerResponse, init?: ResponseInit ⇒ void`

Writes text delta output to a Node.js response-like object. It sets a `Content-Type` header to `text/plain; charset=utf-8` and writes each text delta as a separate chunk.

`ResponseInit`

`status?: number`

The response status code.

`statusText?: string`

The response status text.

`headers?: Record<string, string>`

The response headers.

`toDataStream: (options?: ToDataStreamOptions) => Response`

Converts the result to a data stream.

`ToDataStreamOptions`

`data?: StreamData`

The stream data object.

`getErrorMessage?: (error: unknown) => string`

A function to get the error message from the error object. By default, all errors are masked as "" for safety reasons.

`sendUsage?: boolean`

Whether to send the usage information in the stream. Defaults to true.

`toDataStreamResponse: (options?: ToDataStreamResponseOptions) => Response`

Converts the result to a streamed response object with a stream data part stream. It can be used with the `useChat` and `useCompletion` hooks.

`ToDataStreamResponseOptions`

`status?: number`

The response status code.

`statusText?: string`

The response status text.

`headers?: Record<string, string>`

The response headers.

`data?: StreamData`

The stream data object.

`getErrorMessage?: (error: unknown) => string`

A function to get the error message from the error object. By default, all errors are masked as "" for safety reasons.

`sendUsage?: boolean`

Whether to send the usage information in the stream. Defaults to true.

`toTextStreamResponse: (init?: ResponseInit) => Response`

Creates a simple text stream response. Each text delta is encoded as UTF-8 and sent as a separate chunk. Non-text-delta events are ignored.

`ResponseInit`

`status?: number`

The response status code.

`statusText?: string`

The response status text.

`headers?: Record<string, string>`

The response headers.

Examples

- Learn to stream text generated by a language model in Next.js ↗
- Learn to stream chat completions generated by a language model in Next.js ↗
- Learn to stream text generated by a language model in Node.js ↗
- Learn to stream chat completions generated by a language model in Node.js ↗

Previous
generateText

Next
generateObject