Chatbot

```
gradio.Chatbot(...)
```

Description

Displays a chatbot output showing both user submitted messages and responses. Supports a subset of Markdown including bold, italics, code, tables. Also supports audio/video/image files, which are displayed in the Chatbot, and other kinds of files which are displayed as links.

Behavior

As input: passes the messages in the Chatbot as a `List[List[str | None | Tuple]]`, i.e. a list of lists. The inner list has 2 elements: the user message and the response message. See `Postprocessing` for the format of these messages.

As output: expects function to return a `List[List[str | None | Tuple]]`, i.e. a list of lists. The inner list should have 2 elements: the user message and the response message. The individual messages can be (1) strings in valid Markdown, (2) tuples if sending files: (a filepath or URL to a file, [optional string alt text]) -- if the file is image/video/audio, it is displayed in the Chatbot, or (3) None, in which case the message is not displayed.

Initialization

Parameter	Description
`value` *list[list[str | tuple[str] | tuple[str | Path, str] | None]] | Callable | None* **default: None**	Default value to show in chatbot. If callable, the function will be called whenever the app loads to set the initial value of the component.

ımeter	Description
`label` *str* | *None* **default: None**	The label for this component. Appears above the component and is also used as the header if there are a table of examples for this component. If None and used in a `gr.Interface`, the label will be the name of the parameter this component is assigned to.
`every` *float* | *None* **default: None**	If `value` is a callable, run the function 'every' number of seconds while the client connection is open. Has no effect otherwise. Queue must be enabled. The event can be accessed (e.g. to cancel it) via this component's .load_event attribute.
`show_label` *bool* | *None* **default: None**	if True, will display label.
`container` *bool* **default: True**	If True, will place the component in a container - providing some extra padding around the border.
`scale` *int* | *None* **default: None**	relative width compared to adjacent Components in a Row. For example, if Component A has scale=2, and Component B has scale=1, A will be twice as wide as B. Should be an integer.
`min_width` *int* **default: 160**	minimum pixel width, will wrap if not sufficient screen space to satisfy this value. If a certain scale value results in this Component being narrower than min_width, the min_width parameter will be respected first.
`visible` *bool* **default: True**	If False, component will be hidden.
`elem_id` *str* | *None* **default: None**	An optional string that is assigned as the id of this component in the HTML DOM. Can be used for targeting CSS styles.

ımeter	Description
`elem_classes` *list[str] | str | None* **default: None**	An optional list of strings that are assigned as the classes of this component in the HTML DOM. Can be used for targeting CSS styles.
`render` *bool* **default: True**	If False, component will not render be rendered in the Blocks context. Should be used if the intention is to assign event listeners now but render the component later.
`height` *int | str | None* **default: None**	The height of the component, specified in pixels if a number is passed, or in CSS units if a string is passed.
`latex_delimiters` *list[dict[str, str | bool]] | None* **default: None**	A list of dicts of the form `"left": open delimiter (str), "right": close delimiter (str), "display": whether to display in newline (bool)` that will be used to render LaTeX expressions. If not provided, `latex_delimiters` is set to `[ "left": "$$", "right": "$$", "display": True ]`, so only expressions enclosed in $$ delimiters will be rendered as LaTeX, and in a new line. Pass in an empty list to disable LaTeX rendering. For more information, see the [KaTeX documentation](https://katex.org/docs/autorender.html).
`rtl` *bool* **default: False**	If True, sets the direction of the rendered text to right-to-left. Default is False, which renders text left-to-right.
`show_share_button` *bool | None* **default: None**	If True, will show a share icon in the corner of the component that allows user to share outputs to Hugging Face Spaces Discussions. If False, icon does not appear. If set to None (default behavior), then the icon appears if this Gradio app is launched on Spaces, but not otherwise.
`show_copy_button` *bool* **default: False**	If True, will show a copy button for each chatbot message.

ımeter	Description
`avatar_images` *tuple[str | Path | None, str | Path | None] | None* **default: None**	Tuple of two avatar image paths or URLs for user and bot (in that order). Pass None for either the user or bot image to skip. Must be within the working directory of the Gradio app or an external URL.
`sanitize_html` *bool* **default: True**	If False, will disable HTML sanitization for chatbot messages. This is not recommended, as it can lead to security vulnerabilities.
`render_markdown` *bool* **default: True**	If False, will disable Markdown rendering for chatbot messages.
`bubble_full_width` *bool* **default: True**	If False, the chat bubble will fit to the content of the message. If True (default), the chat bubble will be the full width of the component.
`line_breaks` *bool* **default: True**	If True (default), will enable Github-flavored Markdown line breaks in chatbot messages. If False, single new lines will be ignored. Only applies if `render_markdown` is True.
`likeable` *bool* **default: False**	Whether the chat messages display a like or dislike button. Set automatically by the .like method but has to be present in the signature for it to show up in the config.
`layout` *Literal[('panel', 'bubble')] | None* **default: None**	If "panel", will display the chatbot in a llm style layout. If "bubble", will display the chatbot with message bubbles, with the user and bot messages on alterating sides. Will default to "bubble".

Shortcuts

Class	Interface String Shortcut	Initialization
`gradio.Chatbot`	"chatbot"	Uses default values

os

chatbot_simple | chatbot_multimodal

```
import gradio as gr
import random
import time

with gr.Blocks() as demo:
    chatbot = gr.Chatbot()
    msg = gr.Textbox()
    clear = gr.ClearButton([msg, chatbot])

    def respond(message, chat_history):
        bot_message = random.choice(["How are you?", "I love you", "I'm very hungry"])
```

Event Listeners

Description

Event listeners allow you to capture and respond to user interactions with the UI components you've defined in a Gradio Blocks app. When a user interacts with an element, such as changing a slider value or uploading an image, a function is called.

Supported Event Listeners

The `Chatbot` component supports the following event listeners. Each event listener takes the same parameters, which are listed in the Event Arguments table below.

Listener	Description
`gradio.Chatbot.change(fn, ···)`	Triggered when the value of the Chatbot changes either because of user input (e.g. a user types in a textbox) OR because of a function update (e.g. an image receives a value from the output of an event trigger). See `.input()` for a listener that is only triggered by user input.
`gradio.Chatbot.select(fn, ···)`	Event listener for when the user selects or deselects the Chatbot. Uses event data gradio.SelectData to carry `value` referring to the label of the Chatbot, and `selected` to refer to state of the Chatbot. See EventData documentation on how to use this event data

Listener	Description
`gradio.Chatbot.like(fn, ···)`	This listener is triggered when the user likes/dislikes from within the Chatbot. This event has EventData of type gradio.LikeData that carries information, accessible through LikeData.index and LikeData.value. See EventData documentation on how to use this event data.

Event Arguments

Parameter	Description
`fn` *Callable* | *None* | *Literal['decorator']* **default: "decorator"**	the function to call when this event is triggered. Often a machine learning model's prediction function. Each parameter of the function corresponds to one input component, and the function should return a single value or a tuple of values, with each element in the tuple corresponding to one output component.
`inputs` *Component* | *list[Component]* | *set[Component]* | *None* **default: None**	List of gradio.components to use as inputs. If the function takes no inputs, this should be an empty list.
`outputs` *Component* | *list[Component]* | *None* **default: None**	List of gradio.components to use as outputs. If the function returns no outputs, this should be an empty list.

Parameter	Description
`api_name` *str | None | Literal[False]* **default: None**	defines how the endpoint appears in the API docs. Can be a string, None, or False. If set to a string, the endpoint will be exposed in the API docs with the given name. If None (default), the name of the function will be used as the API endpoint. If False, the endpoint will not be exposed in the API docs and downstream apps (including those that `gr.load` this app) will not be able to use this event.
`scroll_to_output` *bool* **default: False**	If True, will scroll to output component on completion
`show_progress` *Literal[('full', 'minimal', 'hidden')]* **default: "full"**	If True, will show progress animation while pending
`queue` *bool | None* **default: None**	If True, will place the request on the queue, if the queue has been enabled. If False, will not put this event on the queue, even if the queue has been enabled. If None, will use the queue setting of the gradio app.
`batch` *bool* **default: False**	If True, then the function should process a batch of inputs, meaning that it should accept a list of input values for each parameter. The lists should be of equal length (and be up to length `max_batch_size`). The function is then ***required*** to return a tuple of lists (even if there is only 1 output component), with each list in the tuple corresponding to one output component.
`max_batch_size` *int* **default: 4**	Maximum number of inputs to batch together if this is called from the queue (only relevant if batch=True)

Parameter	Description
`preprocess` *bool* **default: True**	If False, will not run preprocessing of component data before running 'fn' (e.g. leaving it as a base64 string if this method is called with the `Image` component).
`postprocess` *bool* **default: True**	If False, will not run postprocessing of component data before returning 'fn' output to the browser.
`cancels` *dict[str, Any] | list[dict[str, Any]] | None* **default: None**	A list of other events to cancel when this listener is triggered. For example, setting cancels=[click_event] will cancel the click_event, where click_event is the return value of another components .click method. Functions that have not yet run (or generators that are iterating) will be cancelled, but functions that are currently running will be allowed to finish.
`every` *float | None* **default: None**	Run this event 'every' number of seconds while the client connection is open. Interpreted in seconds. Queue must be enabled.
`trigger_mode` *Literal[('once', 'multiple', 'always_last')] | None* **default: None**	If "once" (default for all events except `.change()`) would not allow any submissions while an event is pending. If set to "multiple", unlimited submissions are allowed while pending, and "always_last" (default for `.change()` event) would allow a second submission after the pending event is complete.
`js` *str | None* **default: None**	Optional frontend js method to run before running 'fn'. Input arguments for js method are values of 'inputs' and 'outputs', return should be a list of values for output components.

Parameter	Description
`concurrency_limit` *int* | *None* | *Literal['default']* **default: "default"**	If set, this is the maximum number of this event that can be running simultaneously. Can be set to None to mean no concurrency_limit (any number of this event can be running simultaneously). Set to "default" to use the default concurrency limit (defined by the `default_concurrency_limit` parameter in `Blocks.queue()`, which itself is 1 by default).
`concurrency_id` *str* | *None* **default: None**	If set, this is the id of the concurrency group. Events with the same concurrency_id will be limited by the lowest set concurrency_limit.
`show_api` *bool* **default: True**	whether to show this event in the "view API" page of the Gradio app, or in the ".view_api()" method of the Gradio clients. Unlike setting api_name to False, setting show_api to False will still allow downstream apps to use this event. If fn is None, show_api will automatically be se