← UploadButton load →

Video

`gradio.Video(···)`

Description

Creates a video component that can be used to upload/record videos (as an input) or display videos (as an output). For the video to be playable in the browser it must have a compatible container and codec combination. Allowed combinations are .mp4 with h264 codec, .ogg with theora codec, and .webm with vp9 codec. If the component detects that the output video would not be playable in the browser it will attempt to convert it to a playable mp4 video. If the conversion fails, the original video is returned.

Behavior

As input: passes the uploaded video as a `str` filepath or URL whose extension can be modified by `format`.

As output: expects a `str` or `pathlib.Path` filepath to a video which is displayed, or a `Tuple[str | pathlib.Path, str | pathlib.Path | None]` where the first element is a filepath to a video and the second element is an optional filepath to a subtitle file.

Initialization

Parameter	Description
`value` *str | Path | tuple[str | Path, str | Path | None] | Callable | None* **default: None**	A path or URL for the default value that Video component is going to take. Can also be a tuple consisting of (video filepath, subtitle filepath). If a subtitle file is provided, it should be of type .srt or .vtt. Or can be callable, in which case the function will be called whenever the app loads to set the initial value of the component.

meter	Description
`format` *str | None* **default: None**	Format of video format to be returned by component, such as 'avi' or 'mp4'. Use 'mp4' to ensure browser playability. If set to None, video will keep uploaded format.
`sources` *list[Literal[('upload', 'webcam')]] | None* **default: None**	A list of sources permitted for video. "upload" creates a box where user can drop an video file, "webcam" allows user to record a video from their webcam. If None, defaults to ["upload, "webcam"].
`height` *int | str | None* **default: None**	The height of the displayed video, specified in pixels if a number is passed, or in CSS units if a string is passed.
`width` *int | str | None* **default: None**	The width of the displayed video, specified in pixels if a number is passed, or in CSS units if a string is passed.
`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.

ımeter	Description
`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.
`interactive` *bool | None* **default: None**	if True, will allow users to upload a video; if False, can only be used to display videos. If not provided, this is inferred based on whether the component is used as an input or output.
`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.
`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.
`mirror_webcam` *bool* **default: True**	If True webcam will be mirrored. Default is True.

meter	Description
`include_audio` *bool | None* **default: None**	Whether the component should record/retain the audio track for a video. By default, audio is excluded for webcam videos and included for uploaded videos.
`autoplay` *bool* **default: False**	Whether to automatically play the video when the component is used as an output. Note: browsers will not autoplay video files if the user has not interacted with the page yet.
`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_download_button` *bool | None* **default: None**	If True, will show a download icon in the corner of the component that allows user to download the output. If False, icon does not appear. By default, it will be True for output components and False for input components.
`min_length` *int | None* **default: None**	The minimum length of video (in seconds) that the user can pass into the prediction function. If None, there is no minimum length.
`max_length` *int | None* **default: None**	The maximum length of video (in seconds) that the user can pass into the prediction function. If None, there is no maximum length.

Shortcuts

Class	Interface String Shortcut	Initialization
`gradio.Video`	"video"	Uses default values
`gradio.PlayableVideo`	"playablevideo"	Uses format="mp4"

Demos

eo_identity video_subtitle

```python
import gradio as gr
import os


def video_identity(video):
    return video


demo = gr.Interface(video_identity,
                    gr.Video(),
                    "playable_video",
```

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 `Video` 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.Video.change(fn, ···)`	Triggered when the value of the Video 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.Video.clear(fn, ···)`	This listener is triggered when the user clears the Video using the X button for the component.
`gradio.Video.start_recording(fn, ···)`	This listener is triggered when the user starts recording with the Video.

Listener	Description
`gradio.Video.stop_recording(fn, ···)`	This listener is triggered when the user stops recording with the Video.
`gradio.Video.stop(fn, ···)`	This listener is triggered when the user reaches the end of the media playing in the Video.
`gradio.Video.play(fn, ···)`	This listener is triggered when the user plays the media in the Video.
`gradio.Video.pause(fn, ···)`	This listener is triggered when the media in the Video stops for any reason.
`gradio.Video.end(fn, ···)`	This listener is triggered when the user reaches the end of the media playing in the Video.
`gradio.Video.upload(fn, ···)`	This listener is triggered when the user uploads a file into the Video.

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*	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 s' to False will still allow downstream apps to use this event. If fn is None, show_api will automatically be set to False.

← UploadButton

load →