Tools

Smolagents is an experimental API which is subject to change at any time. Results returned by the agents can vary as the APIs or underlying models are prone to change.

To learn more about agents and tools make sure to read the introductory guide. This page contains the API docs for the underlying classes.

Tools

load_tool

smolagents.load_tool

<source>

(repo_idmodel_repo_id: typing.Optional[str] = Nonetoken: typing.Optional[str] = Nonetrust_remote_code: bool = False**kwargs)

Parameters

- **repo_id** (str) — Repo ID of a tool on the Hub.
- **model_repo_id** (str, *optional*) — Use this argument to use a different model than the default one for the tool you selected.
- **token** (str, *optional*) — The token to identify you on hf.co. If unset, will use the token generated when running huggingface-cli login (stored in ~/.huggingface).
- **trust_remote_code** (bool, *optional*, defaults to False) — This needs to be accepted in order to load a tool from Hub.
- **kwargs** (additional keyword arguments, *optional*) — Additional keyword arguments that will be split in two: all arguments relevant to the Hub (such as cache_dir, revision, subfolder) will be used when downloading the files for your tool, and the others will be passed along to its init.

Main function to quickly load a tool from the Hub.

Loading a tool means that you'll download the tool and execute it locally. ALWAYS inspect the tool you're downloading before loading it within your runtime, as you would do when installing a package using pip/npm/apt.

tool

smolagents.tool

<source>

(tool_function: typing.Callable)

Parameters

- **tool_function** — Your function. Should have type hints for each input and a type hint for the output.
- **Should** also have a docstring description including an 'Args —' part where each argument is described.

Converts a function into an instance of a Tool subclass.

Tool

class smolagents.Tool

<source>

(*args**kwargs)

A base class for the functions used by the agent. Subclass this and implement
the forward method as well as the following class attributes:

- **description** (str) — A short description of what your tool does, the inputs it expects and the output(s) it will return. For instance 'This is a tool that downloads a file from a url. It takes the url as input, and returns the text contained in the file'.
- **name** (str) — A performative name that will be used for your tool in the prompt to the agent. For instance "text-classifier" or "image_generator".
- **inputs** (Dict[str, Dict[str, Union[str, type]]]) — The dict of modalities expected for the inputs. It has one typekey and a descriptionkey. This is used by launch_gradio_demo or to make a nice space from your tool, and also can be used in the generated description for your tool.
- **output_type** (type) — The type of the tool output. This is used by launch_gradio_demo or to make a nice space from your tool, and also can be used in the generated description for your tool.

You can also override the method setup() if your tool has an expensive operation to perform before being usable (such as loading a model). setup() will be called the first time you use your tool, but not at instantiation.

from_gradio

<source>

(gradio_tool)

Creates a Tool from a gradio tool.

from_hub

<source>

(repo_id: strtoken: typing.Optional[str] = Nonetrust_remote_code: bool = False**kwargs)

Parameters

- **repo_id** (str) — The name of the repo on the Hub where your tool is defined.
- **token** (str, *optional*) — The token to identify you on hf.co. If unset, will use the token generated when running huggingface-cli login (stored in ~/.huggingface).
- **trust_remote_code(str,** *optional*, defaults to False) — This flags marks that you understand the risk of running remote code and that you trust this tool. If not setting this to True, loading the tool from Hub will fail.
- **kwargs** (additional keyword arguments, *optional*) — Additional keyword arguments that will be split in two: all arguments relevant to the Hub (such as cache_dir, revision, subfolder) will be used when downloading the files for your tool, and the others will be passed along to its init.

Loads a tool defined on the Hub.

Loading a tool from the Hub means that you'll download the tool and execute it locally. ALWAYS inspect the tool you're downloading before loading it within your runtime, as you would do when installing a package using pip/npm/apt.

from_langchain

<source>

(langchain_tool)

Creates a Tool from a langchain tool.

from_space

<source>

(space_id: strname: strdescription: strapi_name: typing.Optional[str] = Nonetoken: typing.Optional[str] = None) → Tool

Parameters

- **space_id** (str) — The id of the Space on the Hub.
- **name** (str) — The name of the tool.
- **description** (str) — The description of the tool.
- **api_name** (str, *optional*) — The specific api_name to use, if the space has several tabs. If not precised, will default to the first available api.

- **token** (str, *optional*) — Add your token to access private spaces or increase your GPU quotas.

Returns

Tool

The Space, as a tool.

Creates a Tool from a Space given its id on the Hub.

Examples:

Copied

```
>>> image_generator = Tool.from_space(
...     space_id="black-forest-labs/FLUX.1-schnell",
...     name="image-generator",
...     description="Generate an image from a prompt"
... )
>>> image = image_generator("Generate an image of a cool surfer in Tahiti")
```

Copied

```
>>> face_swapper = Tool.from_space(
...     "tuan2308/face-swap",
...     "face_swapper",
...     "Tool that puts the face shown on the first image on the second image. You can give it paths to images.",
... )
>>> image = face_swapper('./aymeric.jpeg', './ruth.jpg')
```

push_to_hub

<source>

(repo_id: strcommit_message: str = 'Upload tool'private: typing.Optional[bool] = Nonetoken: typing.Union[bool, str, NoneType] = Nonecreate_pr: bool = False)

Parameters

- **repo_id** (str) — The name of the repository you want to push your tool to. It should contain your organization name when pushing to a given organization.

- **commit_message** (str, *optional*, defaults to "Upload tool") — Message to commit while pushing.
- **private** (bool, *optional*) — Whether to make the repo private. If None (default), the repo will be public unless the organization's default is private. This value is ignored if the repo already exists.
- **token** (bool or str, *optional*) — The token to use as HTTP bearer authorization for remote files. If unset, will use the token generated when running huggingface-cli login (stored in ~/.huggingface).
- **create_pr** (bool, *optional*, defaults to False) — Whether or not to create a PR with the uploaded files or directly commit.

Upload the tool to the Hub.

save

<source>

(output_dir: strtool_file_name: str = 'tool'make_gradio_app: bool = True)

Parameters

- **output_dir** (str) — The folder in which you want to save your tool.
- **tool_file_name** (str, *optional*) — The file name in which you want to save your tool.
- **make_gradio_app** (bool, *optional*, defaults to True) — Whether to also export a requirements.txt file and Gradio UI.

Saves the relevant code files for your tool so it can be pushed to the Hub. This will copy the code of your tool in output_dir as well as autogenerate:

- a {tool_file_name}.py file containing the logic for your tool. If you pass make_gradio_app=True, this will also write:
- an app.py file providing a UI for your tool when it is exported to a Space with tool.push_to_hub()
- a requirements.txt containing the names of the modules used by your tool (as detected when inspecting its code)

setup

<source>

( )

Overwrite this method here for any operation that is expensive and needs to be executed before you start using your tool. Such as loading a big model.

to_dict

<source>

( )

Returns a dictionary representing the tool

launch_gradio_demo

smolagents.launch_gradio_demo

<source>

(tool: Tool)

Parameters

- **tool** (Tool) — The tool for which to launch the demo.

Launches a gradio demo for a tool. The corresponding tool class needs to properly implement the class attributes inputs and output_type.

Default tools

PythonInterpreterTool

class smolagents.PythonInterpreterTool

<source>

(*argsauthorized_imports = None**kwargs)

FinalAnswerTool

class smolagents.FinalAnswerTool

<source>

(*args**kwargs)

UserInputTool

class smolagents.UserInputTool

<source>

(*args**kwargs)

DuckDuckGoSearchTool

class smolagents.DuckDuckGoSearchTool

<source>

(max_results = 10**kwargs)

GoogleSearchTool

class smolagents.GoogleSearchTool

<source>

(provider: str = 'serpapi')

VisitWebpageTool

class smolagents.VisitWebpageTool

<source>

(*args**kwargs)

SpeechToTextTool

class smolagents.SpeechToTextTool

<source>

(*args**kwargs)

ToolCollection

class smolagents.ToolCollection

<source>

(tools: typing.List[smolagents.tools.Tool])

Tool collections enable loading a collection of tools in the agent's toolbox.

Collections can be loaded from a collection in the Hub or from an MCP server, see:

- ToolCollection.from_hub()
- ToolCollection.from_mcp()

For example and usage, see: ToolCollection.from_hub() and ToolCollection.from_mcp()

from_hub

<source>

(collection_slug: strtoken: typing.Optional[str] = Nonetrust_remote_code: bool = False) → ToolCollection

Parameters

- **collection_slug** (str) — The collection slug referencing the collection.

- **token** (str, *optional*) — The authentication token if the collection is private.
- **trust_remote_code** (bool, *optional*, defaults to False) — Whether to trust the remote code.

Returns

ToolCollection

A tool collection instance loaded with the tools.

Loads a tool collection from the Hub.

it adds a collection of tools from all Spaces in the collection to the agent's toolbox

[!NOTE] Only Spaces will be fetched, so you can feel free to add models and datasets to your collection if you'd like for this collection to showcase them.

Example:

Copied

```
>>> from smolagents import ToolCollection, CodeAgent


>>> image_tool_collection = ToolCollection.from_hub("huggingface-tools/diffusion-tools-6630bb19a942c2306a2cdb6f")

>>> agent = CodeAgent(tools=[*image_tool_collection.tools], add_base_tools=True)


>>> agent.run("Please draw me a picture of rivers and lakes.")
```

from_mcp

<source>

(server_parameters) → ToolCollection

Parameters

- **server_parameters** (mcp.StdioServerParameters) — The server parameters to use to
- **connect** to the MCP server. —

Returns

ToolCollection

A tool collection instance.

Automatically load a tool collection from an MCP server.

Note: a separate thread will be spawned to run an asyncio event loop handling the MCP server.

Example:

Copied

```
>>> from smolagents import ToolCollection, CodeAgent
>>> from mcp import StdioServerParameters

>>> server_parameters = StdioServerParameters(
>>>     command="uv",
>>>     args=["--quiet", "pubmedmcp@0.1.3"],
>>>     env={"UV_PYTHON": "3.12", **os.environ},
>>> )

>>> with ToolCollection.from_mcp(server_parameters) as tool_collection:
>>>     agent = CodeAgent(tools=[*tool_collection.tools], add_base_tools=True)
>>>     agent.run("Please find a remedy for hangover.")
```

Agent Types

Agents can handle any type of object in-between tools; tools, being completely multimodal, can accept and return text, image, audio, video, among other types. In order to increase compatibility between tools, as well as to correctly render these returns in ipython (jupyter, colab, ipython notebooks, …), we implement wrapper classes around these types.

The wrapped objects should continue behaving as initially; a text object should still behave as a string, an image object should still behave as a PIL.Image.

These types have three specific purposes:

- Calling to_raw on the type should return the underlying object
- Calling to_string on the type should return the object as a string: that can be the string in case of an AgentText but will be the path of the serialized version of the object in other instances
- Displaying it in an ipython kernel should display the object correctly

AgentText

class smolagents.AgentText

<source>

(value)

Text type returned by the agent. Behaves as a string.

AgentImage

class smolagents.AgentImage

<source>

(value)

Image type returned by the agent. Behaves as a PIL.Image.

save

<source>

(output_bytesformat: str = None**params)

Parameters

- **output_bytes** (bytes) — The output bytes to save the image to.
- **format** (str) — The format to use for the output image. The format is the same as in PIL.Image.save.
- ****params** — Additional parameters to pass to PIL.Image.save.

Saves the image to a file.

to_raw

<source>

( )

Returns the “raw” version of that object. In the case of an AgentImage, it is a PIL.Image.

to_string

<source>

( )

Returns the stringified version of that object. In the case of an AgentImage, it is a path to the serialized version of the image.

AgentAudio

class smolagents.AgentAudio

<source>

(valuesamplerate = 16000)

Audio type returned by the agent.

to_raw

<source>

( )

Returns the “raw” version of that object. It is a torch.Tensor object.

to_string

<source>

( )

Returns the stringified version of that object. In the case of an AgentAudio, it is a path to the serialized version of the audio.