Agents

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.

Agents

Our agents inherit from MultiStepAgent, which means they can act in multiple steps, each step consisting of one thought, then one tool call and execution. Read more in this conceptual guide.

We provide two types of agents, based on the main Agent class.

CodeAgent is the default agent, it writes its tool calls in Python code.

ToolCallingAgent writes its tool calls in JSON.

Both require arguments model and list of tools tools at initialization.

Classes of agents

class smolagents.MultiStepAgent

<

source

>

(tools: typing.List[smolagents.tools.Tool]model: typing.Callable[[typing.List[typing.Dict[str, str]]], smolagents.models.ChatMessage]prompt_templates: typing.Optional[smolagents.agents.PromptTemplates] = Nonemax_steps: int = 20tool_parser: typing.Optional[typing.Callable] = Noneadd_base_tools: bool = Falseverbosity_level: LogLevel = <LogLevel.INFO: 1>grammar: typing.Optional[typing.Dict[str, str]] = Nonemanaged_agents: typing.Optional[typing.List] = Nonestep_callbacks: typing.Optional[typing.List[typing.Callable]] = Noneplanning_interval: typing.Optional[int] = Nonename: typing.Optional[str] = Nonedescription: typing.Optional[str] = Noneprovide_run_summary: bool = Falsefinal_answer_checks: typing.Optional[typing.List[typing.Callable]] = None)

Parameters

tools (list[Tool]) — Tools that the agent can use.

model (Callable[[list[dict[str, str]]], ChatMessage]) — Model that will generate the agent's actions.

prompt_templates (PromptTemplates, optional) — Prompt templates.

max_steps (int, default 20) — Maximum number of steps the agent can take to solve the task.

tool_parser (Callable, optional) — Function used to parse the tool calls from the LLM output.

add_base_tools (bool, default False) — Whether to add the base tools to the agent's tools.

verbosity_level (LogLevel, default LogLevel.INFO) — Level of verbosity of the agent's logs.

grammar (dict[str, str], optional) — Grammar used to parse the LLM output.

managed_agents (list, optional) — Managed agents that the agent can call.

step_callbacks (list[Callable], optional) — Callbacks that will be called at each step.

planning_interval (int, optional) — Interval at which the agent will run a planning step.

name (str, optional) — Necessary for a managed agent only - the name by which this agent can be called.

description (str, optional) — Necessary for a managed agent only - the description of this agent.

provide_run_summary (bool, optional) — Whether to provide a run summary when called as a managed agent.

final_answer_checks (list, optional) — List of Callables to run before returning a final answer for checking validity.

Agent class that solves the given task step by step, using the ReAct framework: While the objective is not reached, the agent will perform a cycle of action (given by the LLM) and observation (obtained from the environment).

execute_tool_call

<

source

>

(tool_name: strarguments: typing.Union[typing.Dict[str, str], str])

Parameters

tool_name (str) — Name of the Tool to execute (should be one from self.tools).

arguments (Dict[str, str]) — Arguments passed to the Tool.

Execute tool with the provided input and returns the result. This method replaces arguments with the actual values from the state if they refer to state variables.

extract_action

<

source

>

(model_output: strsplit_token: str)

Parameters

model_output (str) — Output of the LLM

split_token (str) — Separator for the action. Should match the example in the system prompt.

Parse action from the LLM output

from_folder

<

source

>

(folder: typing.Union[str, pathlib.Path]**kwargs)

Parameters

folder (str or Path) — The folder where the agent is saved.

**kwargs — Additional keyword arguments that will be passed to the agent’s init.

Loads an agent from a local folder.

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(bool, 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 agent, and the others will be passed along to its init.

Loads an agent 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.

initialize_system_prompt

<

source

>

( )

To be implemented in child classes

provide_final_answer

<

source

>

(task: strimages: typing.Optional[list[str]]) → str

Parameters

task (str) — Task to perform.

images (list[str], optional) — Paths to image(s).

Returns

str

Final answer to the task.

Provide the final answer to the task, based on the logs of the agent’s interactions.

push_to_hub

<

source

>

(repo_id: strcommit_message: str = 'Upload agent'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 to. It should contain your organization name when pushing to a given organization.

commit_message (str, optional, defaults to "Upload agent") — Message to commit while pushing.

private (bool, optional, defaults to None) — Whether to make the repo private. If None, 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 to create a PR with the uploaded files or directly commit.

Upload the agent to the Hub.

replay

<

source

>

(detailed: bool = False)

Parameters

detailed (bool, optional) — If True, also displays the memory at each step. Defaults to False. Careful: will increase log length exponentially. Use only for debugging.

Prints a pretty replay of the agent’s steps.

run

<

source

>

(task: strstream: bool = Falsereset: bool = Trueimages: typing.Optional[typing.List[str]] = Noneadditional_args: typing.Optional[typing.Dict] = Nonemax_steps: typing.Optional[int] = None)

Parameters

task (str) — Task to perform.

stream (bool) — Whether to run in a streaming way.

reset (bool) — Whether to reset the conversation or keep it going from previous run.

images (list[str], optional) — Paths to image(s).

additional_args (dict, optional) — Any other variables that you want to pass to the agent run, for instance images or dataframes. Give them clear names!

max_steps (int, optional) — Maximum number of steps the agent can take to solve the task. if not provided, will use the agent's default value.

Run the agent for the given task.

Example:

Copied

```
from smolagents import CodeAgent

agent = CodeAgent(tools=[])

agent.run("What is the result of 2 power 3.7384?")
```

save

<

source

>

(output_dir: strrelative_path: typing.Optional[str] = None)

Parameters

output_dir (str) — The folder in which you want to save your tool.

Saves the relevant code files for your agent. This will copy the code of your agent in output_dir as well as autogenerate:

a tools folder containing the logic for each of the tools under tools/{tool_name}.py.

a managed_agents folder containing the logic for each of the managed agents.

an agent.json file containing a dictionary representing your agent.

a prompt.yaml file containing the prompt templates used by your agent.

an app.py file providing a UI for your agent when it is exported to a Space with agent.push_to_hub()

a requirements.txt containing the names of the modules used by your tool (as detected when inspecting its code)

step

<

source

>

(memory_step: ActionStep)

To be implemented in children classes. Should return either None if the step is not final.

to_dict

<

source

>

( )

Converts agent into a dictionary.

visualize

<

source

>

( )

Creates a rich tree visualization of the agent’s structure.

write_memory_to_messages

<

source

>

(summary_mode: typing.Optional[bool] = False)

Reads past llm_outputs, actions, and observations or errors from the memory into a series of messages that can be used as input to the LLM. Adds a number of keywords (such as PLAN, error, etc) to help the LLM.

class smolagents.CodeAgent

<

source

>

(tools: typing.List[smolagents.tools.Tool]model: typing.Callable[[typing.List[typing.Dict[str, str]]], smolagents.models.ChatMessage]prompt_templates: typing.Optional[smolagents.agents.PromptTemplates] = Nonegrammar: typing.Optional[typing.Dict[str, str]] = Noneadditional_authorized_imports: typing.Optional[typing.List[str]] = Noneplanning_interval: typing.Optional[int] = Noneexecutor_type: str = 'local'executor_kwargs: typing.Optional[typing.Dict[str, typing.Any]] = Nonemax_print_outputs_length: typing.Optional[int] = None**kwargs)

Parameters

tools (list[Tool]) — Tools that the agent can use.

model (Callable[[list[dict[str, str]]], ChatMessage]) — Model that will generate the agent's actions.

prompt_templates (PromptTemplates, optional) — Prompt templates.

grammar (dict[str, str], optional) — Grammar used to parse the LLM output.

additional_authorized_imports (list[str], optional) — Additional authorized imports for the agent.

planning_interval (int, optional) — Interval at which the agent will run a planning step.

executor_type (str, default "local") — Which executor type to use between "local", "e2b", or "docker".

executor_kwargs (dict, optional) — Additional arguments to pass to initialize the executor.

max_print_outputs_length (int, optional) — Maximum length of the print outputs.

**kwargs — Additional keyword arguments.

In this agent, the tool calls will be formulated by the LLM in code format, then parsed and executed.

step

<

source

>

(memory_step: ActionStep)

Perform one step in the ReAct framework: the agent thinks, acts, and observes the result. Returns None if the step is not final.

class smolagents.ToolCallingAgent

<

source

>

(tools: typing.List[smolagents.tools.Tool]model: typing.Callable[[typing.List[typing.Dict[str, str]]], smolagents.models.ChatMessage]prompt_templates: typing.Optional[smolagents.agents.PromptTemplates] = Noneplanning_interval: typing.Optional[int] = None**kwargs)

Parameters

tools (list[Tool]) — Tools that the agent can use.

model (Callable[[list[dict[str, str]]], ChatMessage]) — Model that will generate the agent's actions.

prompt_templates (PromptTemplates, optional) — Prompt templates.

planning_interval (int, optional) — Interval at which the agent will run a planning step.

**kwargs — Additional keyword arguments.

This agent uses JSON-like tool calls, using method model.get_tool_call to leverage the LLM engine's tool calling capabilities.

step

<

source

>

(memory_step: ActionStep)

Perform one step in the ReAct framework: the agent thinks, acts, and observes the result. Returns None if the step is not final.

ManagedAgent

This class is deprecated since 1.8.0: now you simply need to pass attributes name and description to a normal agent to make it callable by a manager agent.

stream_to_gradio

smolagents.stream_to_gradio

<

source

>

(agenttask: strreset_agent_memory: bool = Falseadditional_args: typing.Optional[dict] = None)

Runs an agent with the given task and streams the messages from the agent as gradio ChatMessages.

GradioUI

You must have gradio installed to use the UI. Please run pip install smolagents[gradio] if it’s not the case.

class smolagents.GradioUI

<

source

>

(agent: MultiStepAgentfile_upload_folder: str | None = None)

A one-line interface to launch your agent in Gradio

upload_file

<

source

>

(filefile_uploads_logallowed_file_types = None)

Handle file uploads, default allowed types are .pdf, .docx, and .txt

Prompts

class smolagents.PromptTemplates

<

source

>

( )

Parameters

system_prompt (str) — System prompt.

planning (PlanningPromptTemplate) — Planning prompt templates.

managed_agent (ManagedAgentPromptTemplate) — Managed agent prompt templates.

final_answer (FinalAnswerPromptTemplate) — Final answer prompt templates.

Prompt templates for the agent.

class smolagents.PlanningPromptTemplate

<

source

>

( )

Parameters

initial_facts (str) — Initial facts prompt.

initial_plan (str) — Initial plan prompt.

update_facts_pre_messages (str) — Update facts pre-messages prompt.

update_facts_post_messages (str) — Update facts post-messages prompt.

update_plan_pre_messages (str) — Update plan pre-messages prompt.

update_plan_post_messages (str) — Update plan post-messages prompt.

Prompt templates for the planning step.

class smolagents.ManagedAgentPromptTemplate

<

source

>

( )

Parameters

task (str) — Task prompt.

report (str) — Report prompt.

Prompt templates for the managed agent.

class smolagents.FinalAnswerPromptTemplate

<

source

>

( )

Parameters

pre_messages (str) — Pre-messages prompt.

post_messages (str) — Post-messages prompt.