On this page

agentchat.contrib.agent_builder

AgentBuilder Objects

```
class AgentBuilder()
```

AgentBuilder can help user build an automatic task solving process powered by multi-agent system. Specifically, our building pipeline includes initialize and build. In build(), we prompt a LLM to create multiple participant agents, and specify whether this task need programming to solve. User can save the built agents' config by calling save(), and load the saved configs by load(), which can skip the building process.

__init__

```
def __init__(config_file_or_env: Optional[str] = "OAI_CONFIG_LIST",
             config_file_location: Optional[str] = "",
             builder_model: Optional[str] = "gpt-4",
             agent_model: Optional[str] = "gpt-4",
             host: Optional[str] = "localhost",
             endpoint_building_timeout: Optional[int] = 600,
             max_tokens: Optional[int] = 945,
             max_agents: Optional[int] = 5)
```

(These APIs are experimental and may change in the future.)

Arguments:

- `config_file_or_env` - path or environment of the OpenAI api configs.
- `builder_model` - specify a model as the backbone of build manager.
- `agent_model` - specify a model as the backbone of participant agents.
- `host` - endpoint host.
- `endpoint_building_timeout` - timeout for building up an endpoint server.
- `max_tokens` - max tokens for each agent.
- `max_agents` - max agents for each task.

clear_agent

```
def clear_agent(agent_name: str, recycle_endpoint: Optional[bool] = True)
```

Clear a specific agent by name.

Arguments:

- `agent_name` - the name of agent.
- `recycle_endpoint` - trigger for recycle the endpoint server. If true, the endpoint will be recycled when there is no agent depending on.

clear_all_agents

```
def clear_all_agents(recycle_endpoint: Optional[bool] = True)
```

Clear all cached agents.

build

```
def build(building_task: str,
          default_llm_config: Dict,
          coding: Optional[bool] = None,
          code_execution_config: Optional[Dict] = None,
          use_oai_assistant: Optional[bool] = False,
          **kwargs) -> Tuple[List[autogen.ConversableAgent], Dict]
```

Auto build agents based on the building task.

Arguments:

- `building_task` - instruction that helps build manager (gpt-4) to decide what agent should be built.
- `coding` - use to identify if the user proxy (a code interpreter) should be added.
- `code_execution_config` - specific configs for user proxy (e.g., last_n_messages, work_dir, ...).
- `default_llm_config` - specific configs for LLM (e.g., config_list, seed, temperature, ...).
- `use_oai_assistant` - use OpenAI assistant api instead of self-constructed agent.

Returns:

- `agent_list` - a list of agents.
- `cached_configs` - cached configs.

build_from_library

```
def build_from_library(
        building_task: str,
        library_path_or_json: str,
        default_llm_config: Dict,
        coding: Optional[bool] = True,
        code_execution_config: Optional[Dict] = None,
        use_oai_assistant: Optional[bool] = False,
        embedding_model: Optional[str] = None,
        **kwargs) -> Tuple[List[autogen.ConversableAgent], Dict]
```

Build agents from a library. The library is a list of agent configs, which contains the name and system_message for each agent. We use a build manager to decide what agent in that library should be involved to the task.

Arguments:

- `building_task` - instruction that helps build manager (gpt-4) to decide what agent should be built.
- `library_path_or_json` - path or JSON string config of agent library.
- `default_llm_config` - specific configs for LLM (e.g., config_list, seed, temperature, ...).
- `coding` - use to identify if the user proxy (a code interpreter) should be added.
- `code_execution_config` - specific configs for user proxy (e.g., last_n_messages, work_dir, ...).
- `use_oai_assistant` - use OpenAI assistant api instead of self-constructed agent.
- `embedding_model` - a Sentence-Transformers model use for embedding similarity to select agents from library. if None, an openai model will be prompted to select agents. As reference, chromadb use "all-mpnet-base- v2" as default.

Returns:

- `agent_list` - a list of agents.
- `cached_configs` - cached configs.

save

```
def save(filepath: Optional[str] = None) -> str
```

Save building configs. If the filepath is not specific, this function will create a filename by encrypt the building_task string by md5 with "save_config_" prefix, and save config to the local path.

Arguments:

- `filepath` - save path.

Returns:

- `filepath` - path save.

load

```
def load(filepath: Optional[str] = None,
         config_json: Optional[str] = None,
         use_oai_assistant: Optional[bool] = False,
         **kwargs) -> Tuple[List[autogen.ConversableAgent], Dict]
```

Load building configs and call the build function to complete building without calling online LLMs' api.

Arguments:

- `filepath` - filepath or JSON string for the save config.
- `config_json` - JSON string for the save config.

- `use_oai_assistant` - use OpenAI assistant api instead of self-constructed agent.

Returns:

- `agent_list` - a list of agents.
- `cached_configs` - cached configs.

Previous
« teachability

Next
compressible_agent »