agentchat.contrib.capabilities.teachability

Teachability Objects

```
class Teachability(AgentCapability)
```

Teachability uses a vector database to give an agent the ability to remember user teachings, where the user is any caller (human or not) sending messages to the teachable agent. Teachability is designed to be composable with other agent capabilities. To make any conversable agent teachable, instantiate both the agent and the Teachability class, then pass the agent to teachability.add_to_agent(agent). Note that teachable agents in a group chat must be given unique path_to_db_dir values.

__init__

```
def __init__(verbosity: Optional[int] = 0,
             reset_db: Optional[bool] = False,
             path_to_db_dir: Optional[str] = "./tmp/teachable_agent_db",
             recall_threshold: Optional[float] = 1.5,
             max_num_retrievals: Optional[int] = 10,
             llm_config: Optional[Union[Dict, bool]] = None)
```

Arguments:

- `verbosity` *Optional, int* - # 0 (default) for basic info, 1 to add memory operations, 2 for analyzer messages, 3 for memo lists.
- `reset_db` *Optional, bool* - True to clear the DB before starting. Default False.
- `path_to_db_dir` *Optional, str* - path to the directory where this particular agent's DB is stored. Default "./tmp/teachable_agent_db"
- `recall_threshold` *Optional, float* - The maximum distance for retrieved memos, where 0.0 is exact match. Default 1.5. Larger values allow more (but less relevant) memos to be recalled.
- `max_num_retrievals` *Optional, int* - The maximum number of memos to retrieve from the DB. Default 10.
- `llm_config` *dict or False* - llm inference configuration passed to TextAnalyzerAgent. If None, TextAnalyzerAgent uses llm_config from the teachable agent.

add_to_agent

```
def add_to_agent(agent: ConversableAgent)
```

Adds teachability to the given agent.

prepopulate_db

```
def prepopulate_db()
```

Adds a few arbitrary memos to the DB.

process_last_message

```
def process_last_message(text)
```

Appends any relevant memos to the message text, and stores any apparent teachings in new memos. Uses TextAnalyzerAgent to make decisions about memo storage and retrieval.

MemoStore Objects

```
class MemoStore()
```

Provides memory storage and retrieval for a teachable agent, using a vector database. Each DB entry (called a memo) is a pair of strings: an input text and an output text. The input text might be a question, or a task to perform. The output text might be an answer to the question, or advice on how to perform the task. Vector embeddings are currently supplied by Chroma's default Sentence Transformers.

__init__

```
def __init__(verbosity, reset, path_to_db_dir)
```

Arguments:

- verbosity (Optional, int): 1 to print memory operations, 0 to omit them. 3+ to print memo lists.
- path_to_db_dir (Optional, str): path to the directory where the DB is stored.

list_memos

```
def list_memos()
```

Prints the contents of MemoStore.

reset_db

```
def reset_db()
```

Forces immediate deletion of the DB's contents, in memory and on disk.

add_input_output_pair

```
def add_input_output_pair(input_text, output_text)
```

Adds an input-output pair to the vector DB.

get_nearest_memo

```
def get_nearest_memo(query_text)
```

Retrieves the nearest memo to the given query text.

get_related_memos

```
def get_related_memos(query_text, n_results, threshold)
```

Retrieves memos that are related to the given query text within the specified distance threshold.

prepopulate

```
def prepopulate()
```

Adds a few arbitrary examples to the vector DB, just to make retrieval less trivial.

Edit this page

Previous
« agent_capability

Next
agent_builder »

Community

Discord

Twitter