Frequently Asked Questions

Set your API endpoints

There are multiple ways to construct configurations for LLM inference in the `oai` utilities:

- `get_config_list`: Generates configurations for API calls, primarily from provided API keys.
- `config_list_openai_aoai`: Constructs a list of configurations using both Azure OpenAI and OpenAI endpoints, sourcing API keys from environment variables or local files.
- `config_list_from_json`: Loads configurations from a JSON structure, either from an environment variable or a local JSON file, with the flexibility of filtering configurations based on given criteria.
- `config_list_from_models`: Creates configurations based on a provided list of models, useful when targeting specific models without manually specifying each configuration.
- `config_list_from_dotenv`: Constructs a configuration list from a `.env` file, offering a consolidated way to manage multiple API configurations and keys from a single file.

We suggest that you take a look at this notebook for full code examples of the different methods to configure your model endpoints.

Use the constructed configuration list in agents

Make sure the "config_list" is included in the `llm_config` in the constructor of the LLM-based agent. For example,

```
assistant = autogen.AssistantAgent(
    name="assistant",
    llm_config={"config_list": config_list}
)
```

The `llm_config` is used in the `create` function for LLM inference. When `llm_config` is not provided, the agent will rely on other openai settings such as `openai.api_key` or the environment variable `OPENAI_API_KEY`, which can also work when you'd like to use a single endpoint. You can also explicitly specify that by:

```
assistant = autogen.AssistantAgent(name="assistant", llm_config={"api_key": ...})
```

Unexpected keyword argument 'base_url'

In version >=1, OpenAI renamed their `api_base` parameter to `base_url`. So for older versions, use `api_base` but for newer versions use `base_url`.

Can I use non-OpenAI models?

Yes. Please check https://microsoft.github.io/autogen/blog/2023/07/14/Local-LLMs for an example.

Handle Rate Limit Error and Timeout Error

You can set `max_retries` to handle rate limit error. And you can set `timeout` to handle timeout error. They can all be specified in `llm_config` for an agent, which will be used in the OpenAI client for LLM inference. They can be set differently for different clients if they are set in the `config_list`.

- `max_retries` (int): the total number of times allowed for retrying failed requests for a single client.
- `timeout` (int): the timeout (in seconds) for a single client.

Please refer to the documentation for more info.

How to continue a finished conversation

When you call `initiate_chat` the conversation restarts by default. You can use `send` or `initiate_chat(clear_history=False)` to continue the conversation.

How do we decide what LLM is used for each agent? How many agents can be used?

How do we decide how many agents in the group?

Each agent can be customized. You can use LLMs, tools, or humans behind each agent. If you use an LLM for an agent, use the one best suited for its role. There is no limit of the number of agents, but start from a small number like 2, 3. The more capable is the LLM and the fewer roles you need, the fewer agents you need.

The default user proxy agent doesn't use LLM. If you'd like to use an LLM in UserProxyAgent, the use case could be to simulate user's behavior.

The default assistant agent is instructed to use both coding and language skills. It doesn't have to do coding, depending on the tasks. And you can customize the system message. So if you want to use it for coding, use a model that's good at coding.

Why is code not saved as file?

If you are using a custom system message for the coding agent, please include something like: `If you want the user to save the code in a file before executing it, put # filename: <filename> inside the code block as the first line.` in the system message. This line is in the default system message of the `AssistantAgent`.

If the `# filename` doesn't appear in the suggested code still, consider adding explicit instructions such as "save the code to disk" in the initial user message in `initiate_chat`. The `AssistantAgent` doesn't save all the code by default, because there are cases in which one would just like to finish a task without saving the code.

Code execution

We strongly recommend using docker to execute code. There are two ways to use docker:

1. Run AutoGen in a docker container. For example, when developing in GitHub codespace, AutoGen runs in a docker container. If you are not developing in Github codespace, follow instructions here to install and run AutoGen in docker.
2. Run AutoGen outside of a docker, while performing code execution with a docker container. For this option, set up docker and make sure the python package `docker` is installed. When not installed and `use_docker` is omitted in `code_execution_config`, the code will be executed locally (this behavior is subject to change in future).

Enable Python 3 docker image

You might want to override the default docker image used for code execution. To do that set `use_docker` key of `code_execution_config` property to the name of the image. E.g.:

```
user_proxy = autogen.UserProxyAgent(
    name="agent",
    human_input_mode="TERMINATE",
    max_consecutive_auto_reply=10,
    code_execution_config={"work_dir":"_output", "use_docker":"python:3"},
    llm_config=llm_config,
    system_message="""Reply TERMINATE if the task has been solved at full satisfaction.
Otherwise, reply CONTINUE, or the reason why the task is not solved yet."""
)
```

If you have problems with agents running `pip install` or get errors similar to `Error while fetching server API version: ('Connection aborted.', FileNotFoundError(2, 'No such file or directory'))`, you can choose **'python:3'** as image as shown in the code example above and that should solve the problem.

Agents keep thanking each other when using `gpt-3.5-turbo`

When using `gpt-3.5-turbo` you may often encounter agents going into a "gratitude loop", meaning when they complete a task they will begin congratulating and thanking each other in a continuous loop. This is a limitation in the performance of `gpt-3.5-turbo`, in contrast to `gpt-4` which has no problem remembering instructions. This can hinder the experimentation experience when trying to test out your own use case with cheaper models.

A workaround is to add an additional termination notice to the prompt. This acts a "little nudge" for the LLM to remember that they need to terminate the conversation when their task is complete. You can do this by appending a string such as the following to your user input string:

```
prompt = "Some user query"

termination_notice = (
    '\n\nDo not show appreciation in your responses, say only what is necessary. '
    'if "Thank you" or "You\'re welcome" are said in the conversation, then say TERMINATE '
    'to indicate the conversation is finished and this is your last message.'
)

prompt += termination_notice
```

Note: This workaround gets the job done around 90% of the time, but there are occurrences where the LLM still forgets to terminate the

conversation.

ChromaDB fails in codespaces because of old version of sqlite3

(from issue #251)

Code examples that use chromadb (like retrieval) fail in codespaces due to a sqlite3 requirement.

```
>>> import chromadb
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/vscode/.local/lib/python3.10/site-packages/chromadb/__init__.py", line 69, in <module>
    raise RuntimeError(
RuntimeError: Your system has an unsupported version of sqlite3. Chroma requires sqlite3 >= 3.35.0.
Please visit https://docs.trychroma.com/troubleshooting#sqlite to learn how to upgrade.
```

Workaround:

1. `pip install pysqlite3-binary`
2. `mkdir /home/vscode/.local/lib/python3.10/site-packages/google/colab`

Explanation: Per this gist, linked from the official chromadb docs, adding this folder triggers chromadb to use pysqlite3 instead of the default.

How to register a reply function

(from issue #478)

See here https://microsoft.github.io/autogen/docs/reference/agentchat/conversable_agent/#register_reply

For example, you can register a reply function that gets called when `generate_reply` is called for an agent.

```python
def print_messages(recipient, messages, sender, config):
    if "callback" in config and  config["callback"] is not None:
        callback = config["callback"]
        callback(sender, recipient, messages[-1])
    print(f"Messages sent to: {recipient.name} | num messages: {len(messages)}")
    return False, None  # required to ensure the agent communication flow continues

user_proxy.register_reply(
    [autogen.Agent, None],
    reply_func=print_messages,
    config={"callback": None},
)

assistant.register_reply(
    [autogen.Agent, None],
    reply_func=print_messages,
    config={"callback": None},
)
```

In the above, we register a `print_messages` function that is called each time the agent's `generate_reply` is triggered after receiving a message.

How to get last message ?

Refer to https://microsoft.github.io/autogen/docs/reference/agentchat/conversable_agent/#last_message

How to get each agent message ?

Please refer to https://microsoft.github.io/autogen/docs/reference/agentchat/conversable_agent#chat_messages

When using autogen docker, is it always necessary to reinstall modules?

The "use_docker" arg in an agent's code_execution_config will be set to the name of the image containing the change after execution, when the conversation finishes. You can save that image name. For a new conversation, you can set "use_docker" to the saved name of the image to start execution there.

Community

Discord

Twitter