LLM Configuration

In AutoGen, LLM's are a key component required for agents to understand and react. Each agent's access to and configuration of LLM's is defined by the `llm_config` argument in its constructor.

The simplest possible LLM configuration that is configured to use `gpt-4` from OpenAI would be:

```
import os

llm_config = (
    {
        "config_list": [{"model": "gpt-4", "api_key":
os.environ["OPENAI_API_KEY"]}],
    },
)
```

WARNING

It is important to never commit secrets into your code, therefore we read the OpenAI API key from an environment variable.

This `llm_config` can then be passed to an agent's constructor to enable it to use the LLM.

```
import autogen

assistant = autogen.AssistantAgent(name="assistant",
llm_config=llm_config)
```

`config_list`

Different tasks may require different models, and the `config_list` allows specifying the different endpoints and configurations that are to be used. It is a list of dictionaries, each of which contains the following keys depending on the kind of endpoint being used:

OpenAI | Azure OpenAI | Other OpenAI compatible

- `model` (str, required): The identifier of the model to be used, such as 'gpt-4', 'gpt-3.5-turbo'
- `api_key` (str, optional): The API key required for authenticating requests to the model's API endpoint.
- `api_type`: `openai`. This is optional and defaults to `openai`.
- `base_url` (str, optional): The base URL of the API endpoint. This is the root address where API calls are directed
- `tags` (List[str], optional): Tags which can be used for filtering

Example:

```
{
    "model": "gpt-4",
    "api_key": os.environ['OPENAI_API_KEY']
}
```

TIP

By default this will create a model client which assumes an OpenAI API (or compatible) endpoint. To use custom model clients, see **here**.

`OAI_CONFIG_LIST` Pattern

A common pattern used is to define this `config_list` in a file as JSON and then use the following helper function to load it:

```
config_list = autogen.config_list_from_json(
    env_or_file="OAI_CONFIG_LIST",
)

# Then, create the assistant agent with the config list
assistant = autogen.AssistantAgent(name="assistant", llm_config=
{"config_list": config_list})
```

This can be helpful as it keeps all the configuration in one place across different projects or notebooks.

Why is it a list?

Being a list allows you to define multiple models that can be used by the agent. This is useful for a few reasons:

- If one model times out or fails, the agent can try another model.
- Having a single global list of models and filtering it based on certain keys (e.g. name, tag) in order to pass select models into a certain agent (e.g. use cheaper GPT 3.5 for agents solving easier tasks)

How does an agent decide which model to pick out of the list?

An agent uses the very first model available in the “config_list” and makes LLM calls against this model. If the model fail (e.g. API throttling) the agent will retry the request against the 2nd model and so on until prompt completion is received (or throws an error if none of the models successfully completes the request). There’s no implicit/hidden logic inside agents that is used to pick “the best model for the task”. It is developers responsibility to pick the right models and use them with agents.

Config list filtering

As described above the list can be filtered based on certain criteria. This is defined as a dictionary of key to filter on and value to filter by. For example, if you have a list of configs

and you want to select the one with the model “gpt-3.5-turbo” you can use the following filter:

```
filter_dict = {"model": "gpt-3.5-turbo"}
```

This can then be applied to a config list loaded in Python with `filter_config`:

```
config_list = autogen.filter_config(config_list, filter_dict)
```

Or, directly when loading the config list using `config_list_from_json`:

```
config_list =
autogen.config_list_from_json(env_or_file="OAI_CONFIG_LIST",
filter_dict)
```

Other configuration

Besides the `config_list`, there are other parameters that can be used to configure the LLM. These are split between parameters specifically used by Autogen and those passed into the model client.

Autogen specific parameters

All of these are optional.

- `cache` TODO - is this meant to still be used?
- `cache_seed` TODO - is this meant to still be used?
- `allow_format_str_template` TODO
- `context` TODO
- `filter_func`, function which can be used to filter the completions returned by the model

Extra model client parameters

It is also possible to passthrough paramters through to the OpenAI client. Parameters that correspond to the `OpenAI` client or the `OpenAI` completions create API can be supplied.

This is commonly used for things like `temperature`, or `timeout`.

Example

```
llm_config = {
    "config_list": [
        {
            "model": "gpt-4",
            "api_key": os.environ.get("AZURE_OPENAI_API_KEY"),
            "api_type": "azure",
            "base_url": os.environ.get("AZURE_OPENAI_API_BASE"),
            "api_version": "2023-12-01-preview",
        },
        {
            "model": "llama-7B",
            "base_url": "http://127.0.0.1:8080",
            "api_type": "openai",
        },
    ],
    "temperature": 0.9,
    "timeout": 300,
}
```

Other config list loader helpers

- `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_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.

Edit this page