Table of Contents

Quick Start

Please note that the code provided in this page is *purely* for learning purposes and is far from perfect. Remember to null-check all responses!

Breaking Changes Notice

If you've just updated the package, it is recommended to check the *changelogs* for information on breaking changes.

Setup

This plugin only provides abstraction for the chat completion APIs provided by *other* third party libraries. The plugin itself does not contain the code to interact with the Gemini or GPT APIs. The Gemini client relies on *UGemini*, and the OpenAI client relies on *com.openai.unity*.

Authentication is also handled by the third party libraries. See *UGemini: Setup* and *com.openai.unity: Authentication*

Now you can create an instance of the desired API client. For Gemini:

```
using Uralstech.UAI.Abstraction;
using Uralstech.UAI.Abstraction.Providers.Gemini;

IModelClient client = new GeminiModelClient();
```

For OpenAI:

```
using Uralstech.UAI.Abstraction;
using Uralstech.UAI.Abstraction.Providers.OAI;

IModelClient client = new OAIModelClient();
```

Chat

You can chat with the model using `IModelClient.Chat`.

```
using Uralstech.UAI.Abstraction.Chat;

ChatInferenceResult result = await client.Chat(new Message[]
{
    new Message(Role.System, "You are a helpful, friendly assistant."),
    new Message(Role.User, "What is the capital of India?")
});

Debug.Log("Response: " + result.Messages[^1].Content);
```

The method returns a `ChatInferenceResult` object with the generated messages, including function calls and responses. The model's plain-text response will likely be the last message in the array.

If you don't provide a model to use, it uses the model defined in the client's `DefaultModelId`. You can change this by either changing the default model, or by providing the model ID in the method's `model` parameter.

For clients that support the feature, you can also turn of the safety filters of the model by setting `tryRemoveFilters` to `true`.

Function Calling

You can define functions using the `Function` class. They can then be passed into a variant of the `Chat` function to be called by the model.

```
using Uralstech.UAI.Abstraction.Tools;

ChatInferenceResult result = await client.Chat(new Message[]
{
    new Message(Role.System, "You are a helpful, friendly assistant."),
    new Message(Role.User, "Can you print \"Hello, World\" to the console?")
},
new Function[]
{
    new Function(
        nameof(PrintToConsole),
        "Prints text to the console.",
        new Parameter[]
        {
            new Parameter()
            {
                Name = "text",
                Description = "The text to print.",
                Type = ParameterType.String,
            }
        },
        PrintToConsole
    )
});

Debug.Log("Response: " + result.Messages[^1].Content);
```

```
async Awaitable<JObject> PrintToConsole(JToken input)
{
    await Awaitable.MainThreadAsync(); // Just to make this a valid Awaitable.

    Debug.Log(input["query"].ToObject<string>());
    return new JObject()
    {
        ["status"] = "Printed text to console."
    };
}
```