Models

The Agents SDK comes with out-of-the-box support for OpenAI models in two flavors:

- **Recommended**: the `OpenAIResponsesModel`, which calls OpenAI APIs using the new Responses API.
- The `OpenAIChatCompletionsModel`, which calls OpenAI APIs using the Chat Completions API.

Non-OpenAI models

You can use most other non-OpenAI models via the LiteLLM integration. First, install the litellm dependency group:

```
pip install "openai-agents[litellm]"
```

Then, use any of the supported models with the `litellm/` prefix:

```
claude_agent = Agent(model="litellm/anthropic/claude-3-5-sonnet-20240620", ...)
gemini_agent = Agent(model="litellm/gemini/gemini-2.5-flash-preview-04-17", ...)
```

Other ways to use non-OpenAI models

You can integrate other LLM providers in 3 more ways (examples here):

1. `set_default_openai_client` is useful in cases where you want to globally use an instance of `AsyncOpenAI` as the LLM client. This is for cases where the LLM provider has an OpenAI compatible API endpoint, and you can set the `base_url` and `api_key`. See a configurable example in examples/model_providers/custom_example_global.py.
2. `ModelProvider` is at the `Runner.run` level. This lets you say "use a custom model provider for all agents in this run". See a configurable example in examples/model_providers/custom_example_provider.py.
3. `Agent.model` lets you specify the model on a specific Agent instance. This enables you to mix and match different providers for different agents. See a configurable example in examples/model_providers/custom_example_agent.py. An easy way to use most available models is via the LiteLLM integration.

In cases where you do not have an API key from `platform.openai.com`, we recommend disabling tracing via `set_tracing_disabled()`, or setting up a different tracing processor.

Note

In these examples, we use the Chat Completions API/model, because most LLM providers don't yet support the Responses API. If your LLM provider does support it, we recommend using Responses.

Mixing and matching models

Within a single workflow, you may want to use different models for each agent. For example, you could use a smaller, faster model for triage, while using a larger, more capable model for complex tasks. When configuring an `Agent`, you can select a specific model by either:

1. Passing the name of a model.
2. Passing any model name + a `ModelProvider` that can map that name to a Model instance.
3. Directly providing a `Model` implementation.

Note

While our SDK supports both the `OpenAIResponsesModel` and the `OpenAIChatCompletionsModel` shapes, we recommend using a single model shape for each workflow because the two shapes support a different set of features and tools. If your workflow requires mixing and matching model shapes, make sure that all the features you're using are available on both.

```
from agents import Agent, Runner, AsyncOpenAI, OpenAIChatCompletionsModel
import asyncio

spanish_agent = Agent(
    name="Spanish agent",
    instructions="You only speak Spanish.",
    model="o3-mini", 1
)

english_agent = Agent(
    name="English agent",
    instructions="You only speak English",
    model=OpenAIChatCompletionsModel( 2
        model="gpt-4o",
        openai_client=AsyncOpenAI()
    ),
)

triage_agent = Agent(
    name="Triage agent",
```

```
    instructions="Handoff to the appropriate agent based on the language of the
request.",
    handoffs=[spanish_agent, english_agent],
    model="gpt-3.5-turbo",
)

async def main():
    result = await Runner.run(triage_agent, input="Hola, ¿cómo estás?")
    print(result.final_output)
```

1. Sets the name of an OpenAI model directly.
2. Provides a `Model` implementation.

When you want to further configure the model used for an agent, you can pass `ModelSettings`, which provides optional model configuration parameters such as temperature.

```
from agents import Agent, ModelSettings

english_agent = Agent(
    name="English agent",
    instructions="You only speak English",
    model="gpt-4o",
    model_settings=ModelSettings(temperature=0.1),
)
```

Common issues with using other LLM providers

Tracing client error 401

If you get errors related to tracing, this is because traces are uploaded to OpenAI servers, and you don't have an OpenAI API key. You have three options to resolve this:

1. Disable tracing entirely: `set_tracing_disabled(True)`.
2. Set an OpenAI key for tracing: `set_tracing_export_api_key(...)`. This API key will only be used for uploading traces, and must be from platform.openai.com.
3. Use a non-OpenAI trace processor. See the tracing docs.

Responses API support

The SDK uses the Responses API by default, but most other LLM providers don't yet support it. You may see 404s or similar issues as a result. To resolve, you have two options:

1. Call `set_default_openai_api("chat_completions")`. This works if you are setting `OPENAI_API_KEY` and `OPENAI_BASE_URL` via environment vars.

2. Use `OpenAIChatCompletionsModel`. There are examples here.

Structured outputs support

Some model providers don't have support for structured outputs. This sometimes results in an error that looks something like this:

```
BadRequestError: Error code: 400 - {'error': {'message': "'response_format.type' :
value is not one of the allowed values ['text','json_object']", 'type':
'invalid_request_error'}}
```

This is a shortcoming of some model providers - they support JSON outputs, but don't allow you to specify the `json_schema` to use for the output. We are working on a fix for this, but we suggest relying on providers that do have support for JSON schema output, because otherwise your app will often break because of malformed JSON.

Mixing models across providers

You need to be aware of feature differences between model providers, or you may run into errors. For example, OpenAI supports structured outputs, multimodal input, and hosted file search and web search, but many other providers don't support these features. Be aware of these limitations:

- Don't send unsupported `tools` to providers that don't understand them
- Filter out multimodal inputs before calling models that are text-only
- Be aware that providers that don't support structured JSON outputs will occasionally produce invalid JSON.