Installation & Setup

PydanticAI is available on PyPI as `pydantic-ai` so installation is as simple as:

pip

```
pip install pydantic-ai
```

uv

```
uv add pydantic-ai
```

(Requires Python 3.9+)

This installs the `pydantic_ai` package, core dependencies, and libraries required to use the following LLM APIs:

- OpenAI API
- Google VertexAI API for Gemini models
- Groq API

Use with Pydantic Logfire

PydanticAI has an excellent (but completely optional) integration with Pydantic Logfire to help you view and understand agent runs.

To use Logfire with PydanticAI, install `pydantic-ai` or `pydantic-ai-slim` with the `logfire` optional group:

pip

```
pip install 'pydantic-ai[logfire]'
```

uv

```
uv add 'pydantic-ai[logfire]'
```

From there, follow the Logfire setup docs to configure Logfire.

Running Examples

We distributes the `pydantic_ai_examples` directory as a separate PyPI package (`pydantic-ai-examples`) to make examples extremely easy to customize and run.

To install examples, use the `examples` optional group:

pip

```
pip install 'pydantic-ai[examples]'
```

uv

```
uv add 'pydantic-ai[examples]'
```

To run the examples, follow instructions in the examples docs.

Slim Install

If you know which model you're going to use and want to avoid installing superfluous packages, you can use the `pydantic-ai-slim` package.

If you're using just `OpenAIModel`, run:

pip

```
pip install 'pydantic-ai-slim[openai]'
```

uv

```
uv add 'pydantic-ai-slim[openai]'
```

If you're using just `GeminiModel` (Gemini via the `generativelanguage.googleapis.com` API) no extra dependencies are required, run:

pip

```
pip install pydantic-ai-slim
```

uv

```
uv add pydantic-ai-slim
```

If you're using just `VertexAIModel`, run:

pip

```
pip install 'pydantic-ai-slim[vertexai]'
```

uv

```
uv add 'pydantic-ai-slim[vertexai]'
```

To use just `GroqModel`, run:

pip

```
pip install 'pydantic-ai-slim[groq]'
```

uv

```
uv add 'pydantic-ai-slim[groq]'
```

You can install dependencies for multiple models and use cases, for example:

pip

```
pip install 'pydantic-ai-slim[openai,vertexai,logfire]'
```

uv

```
uv add 'pydantic-ai-slim[openai,vertexai,logfire]'
```

Model Configuration

To use hosted commercial models, you need to configure your local environment with the appropriate API keys.

OpenAI

To use OpenAI through their main API, go to platform.openai.com and follow your nose until you find the place to generate an API key.

Environment variable

Once you have the API key, you can set it as an environment variable:

```
export OPENAI_API_KEY='your-api-key'
```

You can then use `OpenAIModel` by name:

openai_model_by_name.py

```
from pydantic_ai import Agent

agent = Agent('openai:gpt-4o')
...
```

Or initialise the model directly with just the model name:

openai_model_init.py

```
from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIModel

model = OpenAIModel('gpt-4o')
agent = Agent(model)
...
```

`api_key` argument

If you don't want to or can't set the environment variable, you can pass it at runtime via the `api_key` argument:

openai_model_api_key.py

```
from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIModel

model = OpenAIModel('gpt-4o', api_key='your-api-key')
agent = Agent(model)
...
```

Custom OpenAI Client

`OpenAIModel` also accepts a custom `AsyncOpenAI` client via the `openai_client` parameter, so you can customise the `organization`, `project`, `base_url` etc. as defined in the OpenAI API docs.

You could also use the `AsyncAzureOpenAI` client to use the Azure OpenAI API.

openai_azure.py

```
from openai import AsyncAzureOpenAI

from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIModel

client = AsyncAzureOpenAI(
    azure_endpoint='...',
    api_version='2024-07-01-preview',
    api_key='your-api-key',
)

model = OpenAIModel('gpt-4o', openai_client=client)
agent = Agent(model)
...
```

Gemini

`GeminiModel` let's you use the Google's Gemini models through their `generativelanguage.googleapis.com` API.

`GeminiModelName` contains a list of available Gemini models that can be used through this interface.

For prototyping only

Google themselves refer to this API as the "hobby" API, I've received 503 responses from it a number of times. The API is easy to use and useful for prototyping and simple demos, but I would not rely on it in production.

If you want to run Gemini models in production, you should use the VertexAI API described below.

To use `GeminiModel`, go to aistudio.google.com and follow your nose until you find the place to generate an API key.

Environment variable

Once you have the API key, you can set it as an environment variable:

```
export GEMINI_API_KEY=your-api-key
```

You can then use `GeminiModel` by name:

gemini_model_by_name.py

```
from pydantic_ai import Agent

agent = Agent('gemini-1.5-flash')
...
```

Or initialise the model directly with just the model name:

gemini_model_init.py

```
from pydantic_ai import Agent
from pydantic_ai.models.gemini import GeminiModel

model = GeminiModel('gemini-1.5-flash')
agent = Agent(model)
...
```

`api_key` argument

If you don't want to or can't set the environment variable, you can pass it at runtime via the `api_key` argument:

gemini_model_api_key.py

```
from pydantic_ai import Agent
from pydantic_ai.models.gemini import GeminiModel

model = GeminiModel('gemini-1.5-flash', api_key='your-api-key')
agent = Agent(model)
...
```

Gemini via VertexAI

To run Google's Gemini models in production, you should use `VertexAIModel` which uses the `*-aiplatform.googleapis.com` API.

`GeminiModelName` contains a list of available Gemini models that can be used through this interface.

This interface has a number of advantages over `generativelanguage.googleapis.com` documented above:

1. The VertexAI API is more reliably and marginally lower latency in our experience.
2. You can purchase provisioned throughput with VertexAI to guarantee capacity.
3. If you're running PydanticAI inside GCP, you don't need to set up authentication, it should "just work".
4. You can decide which region to use, which might be important from a regulatory perspective, and might improve latency.

The big disadvantage is that for local development you may need to create and configure a "service account", which I've found extremely painful to get right in the past.

Whichever way you authenticate, you'll need to have VertexAI enabled in your GCP account.

application default credentials

Luckily if you're running PydanticAI inside GCP, or you have the `gcloud` CLI installed and configured, you should be able to use `VertexAIModel` without any additional setup.

To use `VertexAIModel`, with application default credentials configured (e.g. with `gcloud`), you can simply use:

vertexai_application_default_credentials.py

```
from pydantic_ai import Agent
from pydantic_ai.models.vertexai import VertexAIModel

model = VertexAIModel('gemini-1.5-flash')
agent = Agent(model)
...
```

Internally this uses `google.auth.default()` from the `google-auth` package to obtain credentials.

Won't fail until `agent.run()`

Because `google.auth.default()` requires network requests and can be slow, it's not run until you call `agent.run()`. Meaning any configuration or permissions error will only be raised when you try to use the model. To for this check to be run, call `await model.agent_model({}, False, None)`.

You may also need to pass the `project_id` argument to `VertexAIModel` if application default credentials don't set a project, if you pass `project_id` and it conflicts with the project set by application default credentials, an error is raised.

service account

If instead of application default credentials, you want to authenticate with a service account, you'll need to create a service account, add it to your GCP project (note: AFAIK this step is necessary even if you created the service account within the project), give that service account the "Vertex AI Service Agent" role, and download the service account JSON file.

Once you have the JSON file, you can use it thus:

vertexai_service_account.py

```
from pydantic_ai import Agent
from pydantic_ai.models.vertexai import VertexAIModel

model = VertexAIModel(
    'gemini-1.5-flash',
    service_account_file='path/to/service-account.json',
)
agent = Agent(model)
...
```

Customising region

Whichever way you authenticate, you can specify which region requests will be sent to via the `region` argument.

Using a region close to your application can improve latency and might be important from a regulatory perspective.

vertexai_region.py

```
from pydantic_ai import Agent
from pydantic_ai.models.vertexai import VertexAIModel
```

```
model = VertexAIModel('gemini-1.5-flash', region='asia-east1')
agent = Agent(model)
...
```

`VertexAiRegion` contains a list of available regions.

Groq

To use Groq through their API, go to console.groq.com/keys and follow your nose until you find the place to generate an API key.

`GroqModelName` contains a list of available Groq models.

Environment variable

Once you have the API key, you can set it as an environment variable:

```
export GROQ_API_KEY='your-api-key'
```

You can then use `GroqModel` by name:

groq_model_by_name.py

```
from pydantic_ai import Agent

agent = Agent('groq:llama-3.1-70b-versatile')
...
```

Or initialise the model directly with just the model name:

groq_model_init.py

```
from pydantic_ai import Agent
from pydantic_ai.models.groq import GroqModel

model = GroqModel('llama-3.1-70b-versatile')
agent = Agent(model)
...
```

`api_key` argument

If you don't want to or can't set the environment variable, you can pass it at runtime via the `api_key` argument:

groq_model_api_key.py

```
from pydantic_ai import Agent
from pydantic_ai.models.groq import GroqModel

model = GroqModel('llama-3.1-70b-versatile', api_key='your-api-key')
agent = Agent(model)
...
```

Ollama

To use Ollama, you must first download the Ollama client, and then download a model.

You must also ensure the Ollama server is running when trying to make requests to it. For more information, please see the Ollama documentation