(quickstart)=

Quickstart

This guide will help you quickly get started with vLLM to perform:

- [Offline batched inference](#quickstart-offline)
- [Online serving using OpenAI-compatible server](#quickstart-online)

Prerequisites

- OS: Linux
- Python: 3.9 -- 3.12

Installation

If you are using NVIDIA GPUs, you can install vLLM using [pip](https://pypi.org/project/vllm/) directly.

It's recommended to use [uv](https://docs.astral.sh/uv/), a very fast Python environment manager, to create and manage Python environments. Please follow the [documentation](https://docs.astral.sh/uv/#getting-started) to install `uv`. After installing `uv`, you can create a new Python environment and install vLLM using the following commands:

```console
uv venv myenv --python 3.12 --seed
source myenv/bin/activate
uv pip install vllm
```

```

You can also use [conda](https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html) to create and manage Python environments.

```console
conda create -n myenv python=3.12 -y
conda activate myenv
pip install vllm
```

:::{note}
For non-CUDA platforms, please refer [here](#installation-index) for specific instructions on how to install vLLM.
:::

(quickstart-offline)=

Offline Batched Inference

With vLLM installed, you can start generating texts for list of input prompts (i.e. offline batch inferencing). See the example script: <gh-file:examples/offline_inference/basic.py>

The first line of this example imports the classes {class}`~vllm.LLM` and {class}`~vllm.SamplingParams`:
```

- {class}`~vllm.LLM` is the main class for running offline inference with vLLM engine.
- {class}`~vllm.SamplingParams` specifies the parameters for the sampling process.

```python
from vllm import LLM, SamplingParams
```

The next section defines a list of input prompts and sampling parameters for text generation. The [sampling temperature](https://arxiv.org/html/2402.05201v1) is set to `0.8` and the [nucleus sampling probability](https://en.wikipedia.org/wiki/Top-p_sampling) is set to `0.95`. You can find more information about the sampling parameters [here](#sampling-params).

```python
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)
```

The {class}`~vllm.LLM` class initializes vLLM's engine and the [OPT-125M model](https://arxiv.org/abs/2205.01068) for offline inference. The list of supported models can be found [here](#supported-models).

```python
```

llm = LLM(model="facebook/opt-125m")
```

:::{note}
By default, vLLM downloads models from [HuggingFace](https://huggingface.co/). If you would like to use models from [ModelScope](https://www.modelscope.cn), set the environment variable `VLLM_USE_MODELSCOPE` before initializing the engine.
:::

Now, the fun part! The outputs are generated using `llm.generate`. It adds the input prompts to the vLLM engine's waiting queue and executes the vLLM engine to generate the outputs with high throughput. The outputs are returned as a list of `RequestOutput` objects, which include all of the output tokens.

```python
outputs = llm.generate(prompts, sampling_params)

for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

(quickstart-online)=

OpenAI-Compatible Server
```

vLLM can be deployed as a server that implements the OpenAI API protocol. This allows vLLM to be used as a drop-in replacement for applications using OpenAI API.
By default, it starts the server at `http://localhost:8000`. You can specify the address with `--host` and `--port` arguments. The server currently hosts one model at a time and implements endpoints such as [list models](https://platform.openai.com/docs/api-reference/models/list), [create chat completion](https://platform.openai.com/docs/api-reference/chat/completions/create), and [create completion](https://platform.openai.com/docs/api-reference/completions/create) endpoints.

Run the following command to start the vLLM server with the [Qwen2.5-1.5B-Instruct](https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct) model:

```console
vllm serve Qwen/Qwen2.5-1.5B-Instruct
```

:::{note}
By default, the server uses a predefined chat template stored in the tokenizer.
You can learn about overriding it [here](#chat-template).
:::

This server can be queried in the same format as OpenAI API. For example, to list the models:

```console
curl http://localhost:8000/v1/models
```

You can pass in the argument `--api-key` or environment variable `VLLM_API_KEY` to enable the

server to check for API key in the header.

OpenAI Completions API with vLLM

Once your server is started, you can query the model with input prompts:

```console
curl http://localhost:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "Qwen/Qwen2.5-1.5B-Instruct",
        "prompt": "San Francisco is a",
        "max_tokens": 7,
        "temperature": 0
    }'
```

Since this server is compatible with OpenAI API, you can use it as a drop-in replacement for any applications using OpenAI API. For example, another way to query the server is via the `openai` Python package:

```python
from openai import OpenAI

# Modify OpenAI's API key and API base to use vLLM's API server.
openai_api_key = "EMPTY"
openai_api_base = "http://localhost:8000/v1"
```

```
client = OpenAI(
    api_key=openai_api_key,
    base_url=openai_api_base,
)
completion = client.completions.create(model="Qwen/Qwen2.5-1.5B-Instruct",
                                       prompt="San Francisco is a")
print("Completion result:", completion)
```

A more detailed client example can be found here: <gh-file:examples/online_serving/openai_completion_client.py>

OpenAI Chat Completions API with vLLM

vLLM is designed to also support the OpenAI Chat Completions API. The chat interface is a more dynamic, interactive way to communicate with the model, allowing back-and-forth exchanges that can be stored in the chat history. This is useful for tasks that require context or more detailed explanations.

You can use the [create chat completion](https://platform.openai.com/docs/api-reference/chat/completions/create) endpoint to interact with the model:

```console
curl http://localhost:8000/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d '{
```

```
        "model": "Qwen/Qwen2.5-1.5B-Instruct",
        "messages": [
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": "Who won the world series in 2020?"}
        ]
    }'
```

Alternatively, you can use the `openai` Python package:

```python
from openai import OpenAI
# Set OpenAI's API key and API base to use vLLM's API server.
openai_api_key = "EMPTY"
openai_api_base = "http://localhost:8000/v1"

client = OpenAI(
    api_key=openai_api_key,
    base_url=openai_api_base,
)

chat_response = client.chat.completions.create(
    model="Qwen/Qwen2.5-1.5B-Instruct",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Tell me a joke."},
    ]
```

```
)

print("Chat response:", chat_response)
```