(huggingface-integration)=

Integration with HuggingFace

This document describes how vLLM integrates with HuggingFace libraries. We will explain step by step what happens under the hood when we run `vllm serve`.

Let's say we want to serve the popular QWen model by running `vllm serve Qwen/Qwen2-7B`.

1. The `model` argument is `Qwen/Qwen2-7B`. vLLM determines whether this model exists by checking for the corresponding config file `config.json`. See this [code snippet](https://github.com/vllm-project/vllm/blob/10b67d865d92e376956345becafc249d4c3c0ab7/vllm/transformers_utils/config.py#L162-L182) for the implementation. Within this process:

    - If the `model` argument corresponds to an existing local path, vLLM will load the config file directly from this path.
    - If the `model` argument is a HuggingFace model ID consisting of a username and model name, vLLM will first try to use the config file from the HuggingFace local cache, using the `model` argument as the model name and the `--revision` argument as the revision. See [their website](https://huggingface.co/docs/huggingface_hub/en/package_reference/environment_variables#hfhome) for more information on how the HuggingFace cache works.
    - If the `model` argument is a HuggingFace model ID but it is not found in the cache, vLLM will download the config file from the HuggingFace model hub. Refer to [this function](https://github.com/vllm-project/vllm/blob/10b67d865d92e376956345becafc249d4c3c0ab7/vllm/transformers_utils/config.py#L91) for the implementation. The input arguments include the `model` argument as the model name, the `--revision` argument as the revision, and the environment variable `HF_TOKEN` as the token to access the model hub. In our case, vLLM will

download the [config.json](https://huggingface.co/Qwen/Qwen2-7B/blob/main/config.json) file.

2. After confirming the existence of the model, vLLM loads its config file and converts it into a dictionary. See this [code snippet](https://github.com/vllm-project/vllm/blob/10b67d865d92e376956345becafc249d4c3c0ab7/vllm/transformers_utils/config.py#L185-L186) for the implementation.

3. Next, vLLM [inspects](https://github.com/vllm-project/vllm/blob/10b67d865d92e376956345becafc249d4c3c0ab7/vllm/transformers_utils/config.py#L189) the `model_type` field in the config dictionary to [generate](https://github.com/vllm-project/vllm/blob/10b67d865d92e376956345becafc249d4c3c0ab7/vllm/transformers_utils/config.py#190-L216) the config object to use. There are some `model_type` values that vLLM directly supports; see [here](https://github.com/vllm-project/vllm/blob/10b67d865d92e376956345becafc249d4c3c0ab7/vllm/transformers_utils/config.py#L48) for the list. If the `model_type` is not in the list, vLLM will use [AutoConfig.from_pretrained](https://huggingface.co/docs/transformers/en/model_doc/auto#transformers.AutoConfig.from_pretrained) to load the config class, with `model`, `--revision`, and `--trust_remote_code` as the arguments. Please note that:

- HuggingFace also has its own logic to determine the config class to use. It will again use the `model_type` field to search for the class name in the transformers library; see [here](https://github.com/huggingface/transformers/tree/main/src/transformers/models) for the list of supported models. If the `model_type` is not found, HuggingFace will use the `auto_map` field from the config JSON file to determine the class name. Specifically, it is the `AutoConfig` field under `auto_map`. See [DeepSeek](https://huggingface.co/deepseek-ai/DeepSeek-V2.5/blob/main/config.json) for an example.

- The `AutoConfig` field under `auto_map` points to a module path in the model's repository. To create the config class, HuggingFace will import the module and use the `from_pretrained` method to load the config class. This can generally cause arbitrary code execution, so it is only executed when `--trust_remote_code` is enabled.

4. Subsequently, vLLM applies some historical patches to the config object. These are mostly related to RoPE configuration; see [here](https://github.com/vllm-project/vllm/blob/127c07480ecea15e4c2990820c457807ff78a057/vllm/transformers_utils/config.py#L244) for the implementation.

5. Finally, vLLM can reach the model class we want to initialize. vLLM uses the `architectures` field in the config object to determine the model class to initialize, as it maintains the mapping from architecture name to model class in [its registry](https://github.com/vllm-project/vllm/blob/127c07480ecea15e4c2990820c457807ff78a057/vllm/model_executor/models/registry.py#L80). If the architecture name is not found in the registry, it means this model architecture is not supported by vLLM. For `Qwen/Qwen2-7B`, the `architectures` field is `["Qwen2ForCausalLM"]`, which corresponds to the `Qwen2ForCausalLM` class in [vLLM's code](https://github.com/vllm-project/vllm/blob/127c07480ecea15e4c2990820c457807ff78a057/vllm/model_executor/models/qwen2.py#L364). This class will initialize itself depending on various configs.

Beyond that, there are two more things vLLM depends on HuggingFace for.

1. **Tokenizer**: vLLM uses the tokenizer from HuggingFace to tokenize the input text. The tokenizer is loaded using [AutoTokenizer.from_pretrained](https://huggingface.co/docs/transformers/en/model_doc/auto#transformers.AutoTokenizer.from_pretrained) with the `model` argument as the model name and the

`--revision` argument as the revision. It is also possible to use a tokenizer from another model by specifying the `--tokenizer` argument in the `vllm serve` command. Other relevant arguments are `--tokenizer-revision` and `--tokenizer-mode`. Please check HuggingFace's documentation for the meaning of these arguments. This part of the logic can be found in the [get_tokenizer](https://github.com/vllm-project/vllm/blob/127c07480ecea15e4c2990820c457807ff78a057/vllm/transformers_utils/tokenizer.py#L87) function. After obtaining the tokenizer, notably, vLLM will cache some expensive attributes of the tokenizer in [get_cached_tokenizer](https://github.com/vllm-project/vllm/blob/127c07480ecea15e4c2990820c457807ff78a057/vllm/transformers_utils/tokenizer.py#L24).

2. **Model weight**: vLLM downloads the model weight from the HuggingFace model hub using the `model` argument as the model name and the `--revision` argument as the revision. vLLM provides the argument `--load-format` to control what files to download from the model hub. By default, it will try to load the weights in the safetensors format and fall back to the PyTorch bin format if the safetensors format is not available. We can also pass `--load-format dummy` to skip downloading the weights.

- It is recommended to use the safetensors format, as it is efficient for loading in distributed inference and also safe from arbitrary code execution. See the [documentation](https://huggingface.co/docs/safetensors/en/index) for more information on the safetensors format. This part of the logic can be found [here](https://github.com/vllm-project/vllm/blob/10b67d865d92e376956345becafc249d4c3c0ab7/vllm/model_executor/model_loader/loader.py#L385). Please note that:

This completes the integration between vLLM and HuggingFace.

In summary, vLLM reads the config file `config.json`, tokenizer, and model weight from the

HuggingFace model hub or a local directory. It uses the config class from either vLLM, HuggingFace transformers, or loads the config class from the model's repository.