

(reasoning-outputs)=

Reasoning Outputs

vLLM offers support for reasoning models like [DeepSeek R1](https://huggingface.co/deepseek-ai/DeepSeek-R1), which are designed to generate outputs containing both reasoning steps and final conclusions.

Reasoning models return a additional ``reasoning_content`` field in their outputs, which contains the reasoning steps that led to the final conclusion. This field is not present in the outputs of other models.

Supported Models

vLLM currently supports the following reasoning models:

- [DeepSeek R1 series](https://huggingface.co/collections/deepseek-ai/deepseek-r1-678e1e131c0169c0bc89728d) (``deepseek_r1``, which looks for ``<think> ... </think>``)

Quickstart

To use reasoning models, you need to specify the ``--enable-reasoning`` and ``--reasoning-parser`` flags when making a request to the chat completion endpoint. The ``--reasoning-parser`` flag specifies the reasoning parser to use for extracting reasoning content from the model output.

```bash

```
vllm serve deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B \
 --enable-reasoning --reasoning-parser deepseek_r1
...
```

Next, make a request to the model that should return the reasoning content in the response.

```
```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,
)

models = client.models.list()
model = models.data[0].id

# Round 1
messages = [{"role": "user", "content": "9.11 and 9.8, which is greater?"}]
response = client.chat.completions.create(model=model, messages=messages)

reasoning_content = response.choices[0].message.reasoning_content
content = response.choices[0].message.content
```

```
print("reasoning_content:", reasoning_content)

print("content:", content)

...

```

The ``reasoning_content`` field contains the reasoning steps that led to the final conclusion, while the ``content`` field contains the final conclusion.

Streaming chat completions

Streaming chat completions are also supported for reasoning models. The ``reasoning_content`` field is available in the ``delta`` field in [chat completion response chunks](<https://platform.openai.com/docs/api-reference/chat/streaming>).

```
```json
{
 "id": "chatcmpl-123",
 "object": "chat.completion.chunk",
 "created": 1694268190,
 "model": "deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B",
 "system_fingerprint": "fp_44709d6fcb",
 "choices": [
 {
 "index": 0,
 "delta": {
 "role": "assistant",
 "reasoning_content": "is",

```

```

 },
 "logprobs": null,
 "finish_reason": null
}
]
}
...

```

Please note that it is not compatible with the OpenAI Python client library. You can use the `requests` library to make streaming requests.

## How to support a new reasoning model

You can add a new `ReasoningParser` similar to  
`vllm/entrypoints/openai/reasoning\_parsers/deepseek\_r1\_reasoning\_parser.py`.

```
```python
```

```
# import the required packages
```

```
from vllm.entrypoints.openai.reasoning_parsers.abs_reasoning_parsers import (
    ReasoningParser, ReasoningParserManager)
```

```
from vllm.entrypoints.openai.protocol import (ChatCompletionRequest,
                                              DeltaMessage)
```

```
# define a reasoning parser and register it to vllm
```

```
# the name list in register_module can be used
```

```
# in --reasoning-parser.
```

```
@ReasoningParserManager.register_module(["example"])
```

```
class ExampleParser(ReasoningParser):
```

```
    def __init__(self, tokenizer: AnyTokenizer):
```

```
        super().__init__(tokenizer)
```

```
    def extract_reasoning_content_streaming(
```

```
        self,
```

```
        previous_text: str,
```

```
        current_text: str,
```

```
        delta_text: str,
```

```
        previous_token_ids: Sequence[int],
```

```
        current_token_ids: Sequence[int],
```

```
        delta_token_ids: Sequence[int],
```

```
) -> Union[DeltaMessage, None]:
```

```
    """
```

```
    Instance method that should be implemented for extracting reasoning
```

```
    from an incomplete response; for use when handling reasoning calls and
```

```
    streaming. Has to be an instance method because it requires state -
```

```
    the current tokens/diffs, but also the information about what has
```

```
    previously been parsed and extracted (see constructor)
```

```
    """
```

```
    def extract_reasoning_content(
```

```
        self, model_output: str, request: ChatCompletionRequest
```

```
) -> Tuple[Optional[str], Optional[str]]:
```

```
    """
```

```
    Extract reasoning content from a complete model-generated string.
```

Used for non-streaming responses where we have the entire model response available before sending to the client.

Parameters:

`model_output: str`

The model-generated string to extract reasoning content from.

`request: ChatCompletionRequest`

The request object that was used to generate the `model_output`.

Returns:

`Tuple[Optional[str], Optional[str]]`

A tuple containing the reasoning content and the content.

"""

...

After defining the reasoning parser, you can use it by specifying the `--reasoning-parser` flag when making a request to the chat completion endpoint.

```
```bash
```

```
vllm serve <model_tag> \
```

```
--enable-reasoning --reasoning-parser example
```

```
```
```

Limitations

- The reasoning content is only available for online serving's chat completion endpoint (`/v1/chat/completions`).
- It is not compatible with the `[structured_outputs](#structured_outputs)` and `[tool_calling](#tool_calling)` features.
- The reasoning content is not available for all models. Check the model's documentation to see if it supports reasoning.