Qwen / **Qwen3-235B-A22B** like 1.03k Follow Qwen 45.8k

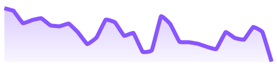
Text Generation · Transformers · Safetensors · qwen3_moe · conversational

arxiv:2309.00071 · arxiv:2505.09388 · License: apache-2.0

Train · Deploy · Use this model

Model card · Files xet · Community 42

Downloads last month
140,979



Safetensors

Model size	235B params
Tensor type	BF16

Chat template · Files info

Inference Providers NEW

Fireworks +2

Text Generation Example

Input a message to start chatting with **Qwen/Qwen3-235B-A22B**.

Your sentence here... Send

View Code Open Playground

Run 15,000+ Models Instantly

Inference Providers let you run inference on thousands of models served by our partners using a simple, unified, OpenAI-compatible serverless API (Learn more).

`Qwen/Qwen3-235B-A22B` is supported by the following Inference Providers:

Together AI · Nscale · Novita · Nebius AI · Fireworks

View API Code Dismiss

Model tree for `Qwen/Qwen3-235B-A22B`

Adapters	9 models
Finetunes	27 models
Quantizations	44 models

Spaces using `Qwen/Qwen3-235B-A22B` 100

Qwen/Qwen3-Demo · umint/ai · yourbench/demo · enzostvs/qwensite · hadadrjt/ai · + 95 Spaces

Collection including `Qwen/Qwen3-235B-A22B`

Qwen3 Collection
84 items • Updated 18 days ago • 1.13k

Qwen3-235B-A22B

Qwen Chat

Qwen3 Highlights

Qwen3 is the latest generation of large language models in Qwen series, offering a comprehensive suite of dense and mixture-of-experts (MoE) models. Built upon extensive training, Qwen3 delivers groundbreaking advancements in reasoning, instruction-following, agent capabilities, and multilingual support, with the following key features:

- **Uniquely support of seamless switching between thinking mode** (for complex logical reasoning, math, and coding) and **non-thinking mode** (for efficient, general-purpose dialogue) **within single model**, ensuring optimal performance across various scenarios.
- **Significantly enhancement in its reasoning capabilities**, surpassing previous QwQ (in thinking mode) and Qwen2.5 instruct models (in non-thinking mode) on mathematics, code generation, and commonsense logical reasoning.
- **Superior human preference alignment**, excelling in creative writing, role-playing, multi-turn dialogues, and instruction following, to deliver a more natural, engaging, and immersive conversational experience.
- **Expertise in agent capabilities**, enabling precise integration with external tools in both thinking and unthinking modes and achieving leading performance among open-source models in complex agent-based tasks.
- **Support of 100+ languages and dialects** with strong capabilities for **multilingual instruction following** and **translation**.

Model Overview

Qwen3-235B-A22B has the following features:

- Type: Causal Language Models
- Training Stage: Pretraining & Post-training
- Number of Parameters: 235B in total and 22B activated
- Number of Paramaters (Non-Embedding): 234B
- Number of Layers: 94
- Number of Attention Heads (GQA): 64 for Q and 4 for KV
- Number of Experts: 128
- Number of Activated Experts: 8
- Context Length: 32,768 natively and 131,072 tokens with YaRN.

For more details, including benchmark evaluation, hardware requirements, and inference performance, please refer to our blog, GitHub, and Documentation.

Quickstart

The code of Qwen3-MoE has been in the latest Hugging Face `transformers` and we advise you to use the latest version of `transformers`.

With `transformers<4.51.0`, you will encounter the following error:

```
KeyError: 'qwen3_moe'
```

The following contains a code snippet illustrating how to use the model generate content based on given inputs.

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model_name = "Qwen/Qwen3-235B-A22B"

# load the tokenizer and the model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype="auto",
    device_map="auto"
)

# prepare the model input
prompt = "Give me a short introduction to large language model."
messages = [
    {"role": "user", "content": prompt}
]
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
    enable_thinking=True # Switches between thinking and non-thinking 
)
model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

# conduct text completion
generated_ids = model.generate(
    **model_inputs,
    max_new_tokens=32768
)
output_ids = generated_ids[0][len(model_inputs.input_ids[0]):].tolist(

# parsing thinking content
try:
    # rindex finding 151668 (</think>)
    index = len(output_ids) - output_ids[::-1].index(151668)
except ValueError:
    index = 0

thinking_content = tokenizer.decode(output_ids[:index], skip_special_t
content = tokenizer.decode(output_ids[index:], skip_special_tokens=Tru

print("thinking content:", thinking_content)
print("content:", content)
```

For deployment, you can use `sglang>=0.4.6.post1` or `vllm>=0.8.5` or to create an OpenAI-compatible API endpoint:

- SGLang:

  ```
  python -m sglang.launch_server --model-path Qwen/Qwen3-235B-A22B -
  ```

- vLLM:

  ```
  vllm serve Qwen/Qwen3-235B-A22B --enable-reasoning --reasoning-pars
  ```

For local use, applications such as Ollama, LMStudio, MLX-LM, llama.cpp, and KTransformers have also supported Qwen3.

Switching Between Thinking and Non-Thinking Mode

> The `enable_thinking` switch is also available in APIs created by SGLang and vLLM. Please refer to our documentation for SGLang and vLLM users.

`enable_thinking=True`

By default, Qwen3 has thinking capabilities enabled, similar to QwQ-32B. This means the model will use its reasoning abilities to enhance the quality of generated responses. For example, when explicitly setting `enable_thinking=True` or leaving it as the default value in `tokenizer.apply_chat_template`, the model will engage its thinking mode.

```
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
    enable_thinking=True  # True is the default value for enable_think
)
```

In this mode, the model will generate think content wrapped in a `<think>...</think>` block, followed by the final response.

> For thinking mode, use `Temperature=0.6`, `TopP=0.95`, `TopK=20`, and `MinP=0` (the default setting in `generation_config.json`). **DO NOT use greedy decoding**, as it can lead to performance degradation and endless repetitions. For more detailed guidance, please refer to the Best Practices section.

`enable_thinking=False`

We provide a hard switch to strictly disable the model's thinking behavior, aligning its functionality with the previous Qwen2.5-Instruct models. This mode is particularly useful in scenarios where disabling thinking is essential for enhancing efficiency.

```
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
    enable_thinking=False  # Setting enable_thinking=False disables th
)
```

In this mode, the model will not generate any think content and will not include a `<think>...</think>` block.

> For non-thinking mode, we suggest using `Temperature=0.7`, `TopP=0.8`, `TopK=20`, and `MinP=0`. For more detailed guidance, please refer to the Best Practices section.

Advanced Usage: Switching Between Thinking and Non-Thinking Modes via User Input

We provide a soft switch mechanism that allows users to dynamically control the model's behavior when `enable_thinking=True`. Specifically, you can add `/think` and `/no_think` to user prompts or system messages to switch the model's thinking mode from turn to turn. The model will follow the most recent instruction in multi-turn conversations.

Here is an example of a multi-turn conversation:

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

class QwenChatbot:
    def __init__(self, model_name="Qwen/Qwen3-235B-A22B"):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(model_name)
        self.history = []

    def generate_response(self, user_input):
        messages = self.history + [{"role": "user", "content": user_inp

        text = self.tokenizer.apply_chat_template(
            messages,
            tokenize=False,
            add_generation_prompt=True
        )

        inputs = self.tokenizer(text, return_tensors="pt")
        response_ids = self.model.generate(**inputs, max_new_tokens=32
        response = self.tokenizer.decode(response_ids, skip_special_to

        # Update history
        self.history.append({"role": "user", "content": user_input})
        self.history.append({"role": "assistant", "content": response}

        return response

# Example Usage
if __name__ == "__main__":
    chatbot = QwenChatbot()

    # First input (without /think or /no_think tags, thinking mode is
    user_input_1 = "How many r's in strawberries?"
    print(f"User: {user_input_1}")
    response_1 = chatbot.generate_response(user_input_1)
    print(f"Bot: {response_1}")
    print("----------------------")

    # Second input with /no_think
    user_input_2 = "Then, how many r's in blueberries? /no_think"
    print(f"User: {user_input_2}")
    response_2 = chatbot.generate_response(user_input_2)
    print(f"Bot: {response_2}")
    print("----------------------")

    # Third input with /think
    user_input_3 = "Really? /think"
    print(f"User: {user_input_3}")
    response_3 = chatbot.generate_response(user_input_3)
    print(f"Bot: {response_3}")
```

> For API compatibility, when `enable_thinking=True`, regardless of whether the user uses `/think` or `/no_think`, the model will always output a block wrapped in `<think>...</think>`. However, the content inside this block may be empty if thinking is disabled. When `enable_thinking=False`, the soft switches are not valid. Regardless of any `/think` or `/no_think` tags input by the user, the model will not generate think content and will not include a `<think>...</think>` block.

Agentic Use

Qwen3 excels in tool calling capabilities. We recommend using Qwen-Agent to make the best use of agentic ability of Qwen3. Qwen-Agent encapsulates tool-calling templates and tool-calling parsers internally, greatly reducing coding complexity.

To define the available tools, you can use the MCP configuration file, use the integrated tool of Qwen-Agent, or integrate other tools by yourself.

```
from qwen_agent.agents import Assistant

# Define LLM
llm_cfg = {
    'model': 'Qwen3-235B-A22B',

    # Use the endpoint provided by Alibaba Model Studio:
    # 'model_type': 'qwen_dashscope',
    # 'api_key': os.getenv('DASHSCOPE_API_KEY'),

    # Use a custom endpoint compatible with OpenAI API:
    'model_server': 'http://localhost:8000/v1',  # api_base
    'api_key': 'EMPTY',

    # Other parameters:
    # 'generate_cfg': {
    #         # Add: When the response content is `<think>this is the
    #         # Do not add: When the response has been separated by rea
    #         'thought_in_content': True,
    #     },
}

# Define Tools
tools = [
    {'mcpServers': {  # You can specify the MCP configuration file
            'time': {
                'command': 'uvx',
                'args': ['mcp-server-time', '--local-timezone=Asia/Shan
            },
            "fetch": {
                "command": "uvx",
                "args": ["mcp-server-fetch"]
            }
        }
    },
  'code_interpreter',  # Built-in tools
]

# Define Agent
bot = Assistant(llm=llm_cfg, function_list=tools)

# Streaming generation
messages = [{'role': 'user', 'content': 'https://qwenlm.github.io/blog/
for responses in bot.run(messages=messages):
    pass
print(responses)
```

Processing Long Texts

Qwen3 natively supports context lengths of up to 32,768 tokens. For conversations where the total length (including both input and output) significantly exceeds this limit, we recommend using RoPE scaling techniques to handle long texts effectively. We have validated the model's performance on context lengths of up to 131,072 tokens using the YaRN method.

YaRN is currently supported by several inference frameworks, e.g., `transformers` and `llama.cpp` for local use, `vllm` and `sglang` for deployment. In general, there are two approaches to enabling YaRN for supported frameworks:

- Modifying the model files: In the `config.json` file, add the `rope_scaling` fields:

  ```
  {
      ...,
      "rope_scaling": {
          "rope_type": "yarn",
          "factor": 4.0,
          "original_max_position_embeddings": 32768
      }
  }
  ```

  For `llama.cpp`, you need to regenerate the GGUF file after the modification.

- Passing command line arguments:

  For `vllm`, you can use

  ```
  vllm serve ... --rope-scaling '{"rope_type":"yarn","factor":4.0,"o
  ```

  For `sglang`, you can use

  ```
  python -m sglang.launch_server ... --json-model-override-args '{"r
  ```

  For `llama-server` from `llama.cpp`, you can use

  ```
  llama-server ... --rope-scaling yarn --rope-scale 4 --yarn-orig-ct
  ```

> If you encounter the following warning
>
> ```
> Unrecognized keys in `rope_scaling` for 'rope_type'='yarn': {'origin
> ```
>
> please upgrade `transformers>=4.51.0`.

> All the notable open-source frameworks implement static YaRN, which means the scaling factor remains constant regardless of input length, **potentially impacting performance on shorter texts.** We advise adding the `rope_scaling` configuration only when processing long contexts is required. It is also recommended to modify the

> `factor` as needed. For example, if the typical context length for your application is 65,536 tokens, it would be better to set `factor` as 2.0.

> The default `max_position_embeddings` in `config.json` is set to 40,960. This allocation includes reserving 32,768 tokens for outputs and 8,192 tokens for typical prompts, which is sufficient for most scenarios involving short text processing. If the average context length does not exceed 32,768 tokens, we do not recommend enabling YaRN in this scenario, as it may potentially degrade model performance.

> The endpoint provided by Alibaba Model Studio supports dynamic YaRN by default and no extra configuration is needed.

Best Practices

To achieve optimal performance, we recommend the following settings:

1. **Sampling Parameters**:
   - For thinking mode (`enable_thinking=True`), use `Temperature=0.6`, `TopP=0.95`, `TopK=20`, and `MinP=0`. **DO NOT use greedy decoding**, as it can lead to performance degradation and endless repetitions.
   - For non-thinking mode (`enable_thinking=False`), we suggest using `Temperature=0.7`, `TopP=0.8`, `TopK=20`, and `MinP=0`.
   - For supported frameworks, you can adjust the `presence_penalty` parameter between 0 and 2 to reduce endless repetitions. However, using a higher value may occasionally result in language mixing and a slight decrease in model performance.
2. **Adequate Output Length**: We recommend using an output length of 32,768 tokens for most queries. For benchmarking on highly complex problems, such as those found in math and programming competitions, we suggest setting the max output length to 38,912 tokens. This provides the model with sufficient space to generate detailed and comprehensive responses, thereby enhancing its overall performance.
3. **Standardize Output Format**: We recommend using prompts to standardize model outputs when benchmarking.
   - **Math Problems**: Include "Please reason step by step, and put your final answer within \boxed{}." in the prompt.
   - **Multiple-Choice Questions**: Add the following JSON structure to the prompt to standardize responses: "Please show your choice in the `answer` field with only the choice letter, e.g., `"answer": "C"`."
4. **No Thinking Content in History**: In multi-turn conversations, the historical model output should only include the final output part and does not need to include the thinking content. It is implemented in the provided chat template in Jinja2. However, for frameworks that do not directly use the Jinja2 chat template, it is up to the developers to ensure that the best practice is followed.

Citation

If you find our work helpful, feel free to give us a cite.

```
@misc{qwen3technicalreport,
      title={Qwen3 Technical Report},
      author={Qwen Team},
      year={2025},
      eprint={2505.09388},
      archivePrefix={arXiv},
      primaryClass={cs.CL},
      url={https://arxiv.org/abs/2505.09388},
}
```