```
!pip install git+https://github.com/shumingma/transformers.git

```

```
Collecting git+https://github.com/shumingma/transformers.git
  Cloning https://github.com/shumingma/transformers.git to /tmp/pip-req-build-gaq_vty7
  Running command git clone --filter=blob:none --quiet https://github.com/shumingma/transformers.git /tmp/pip-req-build-gaq_vty7
  Resolved https://github.com/shumingma/transformers.git to commit 4a01efe84d0120dc2545ff2de445082400d87407
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (0.21.
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers==4.52.0.dev0) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers==4.52.0.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers==4.52.0.dev0) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers==4.52.0.dev0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers==4.52.0.dev0)
Building wheels for collected packages: transformers
  Building wheel for transformers (pyproject.toml) ... done
  Created wheel for transformers: filename=transformers-4.52.0.dev0-py3-none-any.whl size=11413940 sha256=1111ae15305127dd855dc5b8049916
  Stored in directory: /tmp/pip-ephem-wheel-cache-fxk3dmli/wheels/2c/2a/7c/3be0c30fb51a7becc4bcbb536739ae9ed9cc7e633ffbfaf63b
Successfully built transformers
Installing collected packages: transformers
  Attempting uninstall: transformers
    Found existing installation: transformers 4.51.3
    Uninstalling transformers-4.51.3:
      Successfully uninstalled transformers-4.51.3
Successfully installed transformers-4.52.0.dev0
```

```
!git clone https://huggingface.co/microsoft/bitnet-b1.58-2B-4T

```

```
Cloning into 'bitnet-b1.58-2B-4T'...
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 46 (delta 18), reused 0 (delta 0), pack-reused 1 (from 1)
Unpacking objects: 100% (46/46), 2.23 MiB | 5.10 MiB/s, done.
```

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# Specify the model path
model_path = "/content/bitnet-b1.58-2B-4T"

# Load tokenizer (still from the model ID)
tokenizer = AutoTokenizer.from_pretrained("microsoft/bitnet-b1.58-2B-4T")

# Load model from the specified path
model = AutoModelForCausalLM.from_pretrained(
    model_path,
    torch_dtype=torch.bfloat16
)

# ... (rest of your code remains the same)
```

```
tokenizer_config.json: 100%  50.8k/50.8k [00:00<00:00, 957kB/s]
tokenizer.json: 100%  9.09M/9.09M [00:00<00:00, 25.9MB/s]
special_tokens_map.json: 100%  73.0/73.0 [00:00<00:00, 2.07kB/s]
You don't have a GPU available to load the model, the inference will be slow because of weight unpacking
No CUDA runtime is found, using CUDA_HOME='/usr/local/cuda'
```

```
# Apply the chat template
messages = [
    {"role": "system", "content": "You are a helpful AI assistant."},
    {"role": "user", "content": "How are you?"},
```

```
]
prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
chat_input = tokenizer(prompt, return_tensors="pt").to(model.device)

# Generate response
chat_outputs = model.generate(**chat_input, max_new_tokens=11)
response = tokenizer.decode(chat_outputs[0][chat_input['input_ids'].shape[-1]:], skip_special_tokens=True) # Decode only the response pa
print("\nAssistant Response:", response)
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.

Assistant Response: I'm just a computer program, so I don't
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

/content/bitnet-b1.58-2B-4T/configuration_bitnet.py

```
#/content/bitnet-b1.58-2B-4T/configuration_bitnet.py



# coding=utf-8
# Copyright 2022 EleutherAI and the HuggingFace Inc. team. All rights reserved.
#
# This code is based on EleutherAI's GPT-NeoX library and the GPT-NeoX
# and OPT implementations in this library. It has been modified from its
# original forms to accommodate minor architectural differences compared
# to GPT-NeoX and OPT used by the Meta AI team that trained the model.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
""" LLaMA model configuration"""

from transformers.configuration_utils import PretrainedConfig
from transformers.utils import logging


logger = logging.get_logger(__name__)

LLAMA_PRETRAINED_CONFIG_ARCHIVE_MAP = {}


class BitnetConfig(PretrainedConfig):
    r"""
    This is the configuration class to store the configuration of a [`BitnetModel`]. It is used to instantiate an LLaMA
    model according to the specified arguments, defining the model architecture. Instantiating a configuration with the
    defaults will yield a similar configuration to that of the LLaMA-7B.
    Configuration objects inherit from [`PretrainedConfig`] and can be used to control the model outputs. Read the
    documentation from [`PretrainedConfig`] for more information.
    Args:
        vocab_size (`int`, *optional*, defaults to 32000):
            Vocabulary size of the LLaMA model. Defines the number of different tokens that can be represented by the
            `inputs_ids` passed when calling [`BitnetModel`]
        hidden_size (`int`, *optional*, defaults to 4096):
            Dimension of the hidden representations.
        intermediate_size (`int`, *optional*, defaults to 11008):
            Dimension of the MLP representations.
        num_hidden_layers (`int`, *optional*, defaults to 32):
            Number of hidden layers in the Transformer decoder.
        num_attention_heads (`int`, *optional*, defaults to 32):
```

```
            Number of attention heads for each attention layer in the Transformer decoder.
        num_key_value_heads (`int`, *optional*):
            This is the number of key_value heads that should be used to implement Grouped Query Attention. If
            `num_key_value_heads=num_attention_heads`, the model will use Multi Head Attention (MHA), if
            `num_key_value_heads=1 the model will use Multi Query Attention (MQA) otherwise GQA is used. When
            converting a multi-head checkpoint to a GQA checkpoint, each group key and value head should be constructed
            by meanpooling all the original heads within that group. For more details checkout [this
            paper](https://arxiv.org/pdf/2305.13245.pdf). If it is not specified, will default to
            `num_attention_heads`.
        hidden_act (`str` or `function`, *optional*, defaults to `"silu"`):
            The non-linear activation function (function or string) in the decoder.
        max_position_embeddings (`int`, *optional*, defaults to 2048):
            The maximum sequence length that this model might ever be used with. Bitnet 1 supports up to 2048 tokens,
            Bitnet 2 up to 4096, CodeBitnet up to 16384.
        initializer_range (`float`, *optional*, defaults to 0.02):
            The standard deviation of the truncated_normal_initializer for initializing all weight matrices.
        rms_norm_eps (`float`, *optional*, defaults to 1e-06):
            The epsilon used by the rms normalization layers.
        use_cache (`bool`, *optional*, defaults to `True`):
            Whether or not the model should return the last key/values attentions (not used by all models). Only
            relevant if `config.is_decoder=True`.
        pad_token_id (`int`, *optional*):
            Padding token id.
        bos_token_id (`int`, *optional*, defaults to 1):
            Beginning of stream token id.
        eos_token_id (`int`, *optional*, defaults to 2):
            End of stream token id.
        pretraining_tp (`int`, *optional*, defaults to 1):
            Experimental feature. Tensor parallelism rank used during pretraining. Please refer to [this
            document](https://huggingface.co/docs/transformers/main/perf_train_gpu_many#tensor-parallelism) to understand more about it
            necessary to ensure exact reproducibility of the pretraining results. Please refer to [this
            issue](https://github.com/pytorch/pytorch/issues/76232).
        tie_word_embeddings (`bool`, *optional*, defaults to `False`):
            Whether to tie weight embeddings
        rope_theta (`float`, *optional*, defaults to 10000.0):
            The base period of the RoPE embeddings.
        rope_scaling (`Dict`, *optional*):
            Dictionary containing the scaling configuration for the RoPE embeddings. Currently supports two scaling
            strategies: linear and dynamic. Their scaling factor must be a float greater than 1. The expected format is
            `{"type": strategy name, "factor": scaling factor}`. When using this flag, don't update
            `max_position_embeddings` to the expected new maximum. See the following thread for more information on how
            these scaling strategies behave:
            https://www.reddit.com/r/LocalLLaMA/comments/14mrgpr/dynamically_scaled_rope_further_increases/. This is an
            experimental feature, subject to breaking API changes in future versions.
        attention_bias (`bool`, defaults to `False`, *optional*, defaults to `False`):
            Whether to use a bias in the query, key, value and output projection layers during self-attention.
        attention_dropout (`float`, *optional*, defaults to 0.0):
            The dropout ratio for the attention probabilities.
    ```python
    >>> from transformers import BitnetModel, BitnetConfig
    >>> # Initializing a LLaMA llama-7b style configuration
    >>> configuration = BitnetConfig()
    >>> # Initializing a model from the llama-7b style configuration
    >>> model = BitnetModel(configuration)
    >>> # Accessing the model configuration
    >>> configuration = model.config
    ```"""

    model_type = "llama"
    keys_to_ignore_at_inference = ["past_key_values"]

    def __init__(
        self,
        vocab_size=32000,
        hidden_size=4096,
        intermediate_size=11008,
        num_hidden_layers=32,
        num_attention_heads=32,
        num_key_value_heads=None,
        hidden_act="silu",
        max_position_embeddings=2048,
        initializer_range=0.02,
        rms_norm_eps=1e-6,
        use_cache=True,
        pad_token_id=None,
        bos_token_id=1,
        eos_token_id=2,
```

```
        pretraining_tp=1,
        tie_word_embeddings=False,
        rope_theta=10000.0,
        rope_scaling=None,
        attention_bias=False,
        attention_dropout=0.0,
        weight_bits=1,
        input_bits=8,
        **kwargs,
    ):
        self.vocab_size = vocab_size
        self.max_position_embeddings = max_position_embeddings
        self.hidden_size = hidden_size
        self.intermediate_size = intermediate_size
        self.num_hidden_layers = num_hidden_layers
        self.num_attention_heads = num_attention_heads

        # for backward compatibility
        if num_key_value_heads is None:
            num_key_value_heads = num_attention_heads

        self.num_key_value_heads = num_key_value_heads
        self.hidden_act = hidden_act
        self.initializer_range = initializer_range
        self.rms_norm_eps = rms_norm_eps
        self.pretraining_tp = pretraining_tp
        self.use_cache = use_cache
        self.rope_theta = rope_theta
        self.rope_scaling = rope_scaling
        self._rope_scaling_validation()
        self.attention_bias = attention_bias
        self.attention_dropout = attention_dropout
        self.weight_bits = weight_bits
        self.input_bits = input_bits

        super().__init__(
            pad_token_id=pad_token_id,
            bos_token_id=bos_token_id,
            eos_token_id=eos_token_id,
            tie_word_embeddings=tie_word_embeddings,
            **kwargs,
        )

    def _rope_scaling_validation(self):
        """
        Validate the `rope_scaling` configuration.
        """
        if self.rope_scaling is None:
            return

        if not isinstance(self.rope_scaling, dict) or len(self.rope_scaling) != 2:
            raise ValueError(
                "`rope_scaling` must be a dictionary with with two fields, `type` and `factor`, "
                f"got {self.rope_scaling}"
            )
        rope_scaling_type = self.rope_scaling.get("type", None)
        rope_scaling_factor = self.rope_scaling.get("factor", None)
        if rope_scaling_type is None or rope_scaling_type not in ["linear", "dynamic"]:
            raise ValueError(
                f"`rope_scaling`'s type field must be one of ['linear', 'dynamic'], got {rope_scaling_type}"
            )
        if rope_scaling_factor is None or not isinstance(rope_scaling_factor, float) or rope_scaling_factor <= 1.0:
            raise ValueError(f"`rope_scaling`'s factor field must be a float > 1, got {rope_scaling_factor}")
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
from huggingface_hub import HfApi, upload_folder

# إعدادات المستخدم
token = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

```
repo_id = "rakmik/bitnetrun"
folder_path = "/content/bitnet-b1.58-2B-4T"  # ← غير هذا إلى مسار مجلدك المحلي

# رفع المجلد
upload_folder(
    repo_id=repo_id,
    folder_path=folder_path,
    path_in_repo="",  # يمكنك تحديد مجلد داخل الريبو إذا رغبت
    repo_type="model",  # أو "dataset" إذا كان مستودع بيانات
    token=token,
)
print("✅ تم رفع المجلد بنجاح إلى:", repo_id)

```

```
model.safetensors: 100%  1.18G/1.18G [00:29<00:00, 44.6MB/s]
✅ تم رفع المجلد بنجاح إلى: rakmik/bitnetrun
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

model_id = "rakmik/bitnetrun"

# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.bfloat16
)

# Apply the chat template
messages = [
    {"role": "system", "content": "You are a helpful AI assistant."},
    {"role": "user", "content": "How are you?"},
]
prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
chat_input = tokenizer(prompt, return_tensors="pt").to(model.device)

# Generate response
chat_outputs = model.generate(**chat_input, max_new_tokens=11)
response = tokenizer.decode(chat_outputs[0][chat_input['input_ids'].shape[-1]:], skip_special_tokens=True) # Decode only the response par
print("\nAssistant Response:", response)

```

```
tokenizer_config.json: 100%  50.8k/50.8k [00:00<00:00, 1.03MB/s]
tokenizer.json: 100%  9.09M/9.09M [00:00<00:00, 24.3MB/s]
special_tokens_map.json: 100%  73.0/73.0 [00:00<00:00, 6.08kB/s]
config.json: 100%  803/803 [00:00<00:00, 68.3kB/s]
model.safetensors: 92%  1.08G/1.18G [00:30<00:00, 105MB/s]
Error while downloading from https://cdn-lfs-us-1.hf.co/repos/a3/7b/a37bcf8eca6b5df2fea1359acb81db8b715d3c720d5857750b05c83c7892ee12/814
Trying to resume download...
WARNING:huggingface_hub.file_download:Error while downloading from https://cdn-lfs-us-1.hf.co/repos/a3/7b/a37bcf8eca6b5df2fea1359acb81db
Trying to resume download...
model.safetensors: 100%  1.18G/1.18G [00:01<00:00, 109MB/s]
generation_config.json: 100%  199/199 [00:00<00:00, 11.3kB/s]
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.

Assistant Response: As an AI, I don't have feelings, but
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Apply the chat template
messages = [
    {"role": "system", "content": "You are a helpful AI assistant."},
    {"role": "user", "content": "how is python?"},
]
prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
chat_input = tokenizer(prompt, return_tensors="pt").to(model.device)

# Generate response
chat_outputs = model.generate(**chat_input, max_new_tokens=55)
response = tokenizer.decode(chat_outputs[0][chat_input['input_ids'].shape[-1]:], skip_special_tokens=True) # Decode only the response par
print("\nAssistant Response:", response)
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.

Assistant Response: Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used

Here are some of its key features:

1. **Easy to Learn**: Python has a
```

```
Start coding or generate with AI.
```