**Llama2**

**Overview**

The Llama2 model was proposed in [LLaMA: Open Foundation and Fine-Tuned Chat Models](https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/) by Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushka rMishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing EllenTan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, Thomas Scialom. It is a collection of foundation language models ranging from 7B to 70B parameters, with checkpoints finetuned for chat application!

The abstract from the paper is the following:

*In this work, we develop and release Llama 2, a collection of pretrained and fine-tuned large language models (LLMs) ranging in scale from 7 billion to 70 billion parameters. Our fine-tuned LLMs, called Llama 2-Chat, are optimized for dialogue use cases. Our models outperform open-source chat models on most benchmarks we tested, and based on our human evaluations for helpfulness and safety, may be a suitable substitute for closed-source models. We provide a detailed description of our approach to fine-tuning and safety improvements of Llama 2-Chat in order to enable the community to build on our work and contribute to the responsible development of LLMs.*

Checkout all Llama2 model checkpoints [here](https://huggingface.co/models?search=llama2). This model was contributed by [Arthur Zucker](https://huggingface.co/ArthurZ) with contributions from [Lysandre Debut](https://huggingface.co/lysandre). The code of the implementation in Hugging Face is based on GPT-NeoX [here](https://github.com/EleutherAI/gpt-neox). The original code of the authors can be found [here](https://github.com/facebookresearch/llama).

**Usage tips**

The Llama2 models were trained using bfloat16, but the original inference uses float16. The checkpoints uploaded on the Hub use torch\_dtype = 'float16', which will be used by the AutoModel API to cast the checkpoints from torch.float32 to torch.float16.

The dtype of the online weights is mostly irrelevant unless you are using torch\_dtype="auto" when initializing a model using model = AutoModelForCausalLM.from\_pretrained("path", torch\_dtype = "auto"). The reason is that the model will first be downloaded ( using the dtype of the checkpoints online), then it will be casted to the default dtype of torch (becomes torch.float32), and finally, if there is a torch\_dtype provided in the config, it will be used.

Training the model in float16 is not recommended and is known to produce nan; as such, the model should be trained in bfloat16.

Tips:

* Weights for the Llama2 models can be obtained by filling out [this form](https://ai.meta.com/resources/models-and-libraries/llama-downloads/)
* The architecture is very similar to the first Llama, with the addition of Grouped Query Attention (GQA) following this [paper](https://arxiv.org/pdf/2305.13245.pdf)
* Setting config.pretraining\_tp to a value different than 1 will activate the more accurate but slower computation of the linear layers, which should better match the original logits.
* The original model uses pad\_id = -1 which means that there is no padding token. We can’t have the same logic, make sure to add a padding token using tokenizer.add\_special\_tokens({"pad\_token":"<pad>"}) and resize the token embedding accordingly. You should also set the model.config.pad\_token\_id. The embed\_tokens layer of the model is initialized with self.embed\_tokens = nn.Embedding(config.vocab\_size, config.hidden\_size, self.config.padding\_idx), which makes sure that encoding the padding token will output zeros, so passing it when initializing is recommended.
* After filling out the form and gaining access to the model checkpoints, you should be able to use the already converted checkpoints. Otherwise, if you are converting your own model, feel free to use the [conversion script](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/convert_llama_weights_to_hf.py). The script can be called with the following (example) command:

Copied

python src/transformers/models/llama/convert\_llama\_weights\_to\_hf.py \

--input\_dir /path/to/downloaded/llama/weights --model\_size 7B --output\_dir /output/path

* After conversion, the model and tokenizer can be loaded via:

Copied

from transformers import LlamaForCausalLM, LlamaTokenizer

tokenizer = LlamaTokenizer.from\_pretrained("/output/path")

model = LlamaForCausalLM.from\_pretrained("/output/path")

Note that executing the script requires enough CPU RAM to host the whole model in float16 precision (even if the biggest versions come in several checkpoints they each contain a part of each weight of the model, so we need to load them all in RAM). For the 75B model, it’s thus 145GB of RAM needed.

* The LLaMA tokenizer is a BPE model based on [sentencepiece](https://github.com/google/sentencepiece). One quirk of sentencepiece is that when decoding a sequence, if the first token is the start of the word (e.g. “Banana”), the tokenizer does not prepend the prefix space to the string.
* When using Flash Attention 2 via attn\_implementation="flash\_attention\_2", don’t pass torch\_dtype to the from\_pretrained class method and use Automatic Mixed-Precision training. When using Trainer, it is simply specifying either fp16 or bf16 to True. Otherwise, make sure you are using torch.autocast. This is required because the Flash Attention only support fp16 and bf16 data type.

**Resources**

A list of official Hugging Face and community (indicated by 🌎) resources to help you get started with LLaMA2. If you’re interested in submitting a resource to be included here, please feel free to open a Pull Request and we’ll review it! The resource should ideally demonstrate something new instead of duplicating an existing resource.

* [Llama 2 is here - get it on Hugging Face](https://huggingface.co/blog/llama2), a blog post about Llama 2 and how to use it with 🤗 Transformers and 🤗 PEFT.
* [LLaMA 2 - Every Resource you need](https://www.philschmid.de/llama-2), a compilation of relevant resources to learn about LLaMA 2 and how to get started quickly.

Text Generation

* A [notebook](https://colab.research.google.com/drive/1PEQyJO1-f6j0S_XJ8DV50NkpzasXkrzd?usp=sharing) on how to fine-tune Llama 2 in Google Colab using QLoRA and 4-bit precision. 🌎
* A [notebook](https://colab.research.google.com/drive/134o_cXcMe_lsvl15ZE_4Y75Kstepsntu?usp=sharing) on how to fine-tune the “Llama-v2-7b-guanaco” model with 4-bit QLoRA and generate Q&A datasets from PDFs. 🌎

Text Classification

* A [notebook](https://colab.research.google.com/drive/1ggaa2oRFphdBmqIjSEbnb_HGkcIRC2ZB?usp=sharing) on how to fine-tune the Llama 2 model with QLoRa, TRL, and Korean text classification dataset. 🌎🇰🇷

⚗️ Optimization

* [Fine-tune Llama 2 with DPO](https://huggingface.co/blog/dpo-trl), a guide to using the TRL library’s DPO method to fine tune Llama 2 on a specific dataset.
* [Extended Guide: Instruction-tune Llama 2](https://www.philschmid.de/instruction-tune-llama-2), a guide to training Llama 2 to generate instructions from inputs, transforming the model from instruction-following to instruction-giving.
* A [notebook](https://colab.research.google.com/drive/1SYpgFpcmtIUzdE7pxqknrM4ArCASfkFQ?usp=sharing) on how to fine-tune the Llama 2 model on a personal computer using QLoRa and TRL. 🌎

⚡️ Inference

* A [notebook](https://colab.research.google.com/drive/1TC56ArKerXUpbgRy5vM3woRsbTEVNq7h?usp=sharing) on how to quantize the Llama 2 model using GPTQ from the AutoGPTQ library. 🌎
* A [notebook](https://colab.research.google.com/drive/1X1z9Q6domMKl2CnEM0QGHNwidLfR4dW2?usp=sharing) on how to run the Llama 2 Chat Model with 4-bit quantization on a local computer or Google Colab. 🌎

🚀 Deploy

* [Fine-tune LLaMA 2 (7-70B) on Amazon SageMaker](https://www.philschmid.de/sagemaker-llama2-qlora), a complete guide from setup to QLoRA fine-tuning and deployment on Amazon SageMaker.
* [Deploy Llama 2 7B/13B/70B on Amazon SageMaker](https://www.philschmid.de/sagemaker-llama-llm), a guide on using Hugging Face’s LLM DLC container for secure and scalable deployment.

**LlamaConfig**

classtransformers.**LlamaConfig**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/configuration_llama.py#L31)

( vocab\_size = 32000hidden\_size = 4096intermediate\_size = 11008num\_hidden\_layers = 32num\_attention\_heads = 32num\_key\_value\_heads = Nonehidden\_act = 'silu'max\_position\_embeddings = 2048initializer\_range = 0.02rms\_norm\_eps = 1e-06use\_cache = Truepad\_token\_id = Nonebos\_token\_id = 1eos\_token\_id = 2pretraining\_tp = 1tie\_word\_embeddings = Falserope\_theta = 10000.0rope\_scaling = Noneattention\_bias = Falseattention\_dropout = 0.0\*\*kwargs )

**Parameters**

* **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 [LlamaModel](https://huggingface.co/docs/transformers/main/en/model_doc/llama#transformers.LlamaModel)
* **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. Llama 1 supports up to 2048 tokens, Llama 2 up to 4096, CodeLlama 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/parallelism) to understand more about it. This value is 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.

This is the configuration class to store the configuration of a [LlamaModel](https://huggingface.co/docs/transformers/main/en/model_doc/llama#transformers.LlamaModel). 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](https://huggingface.co/docs/transformers/main/en/main_classes/configuration" \l "transformers.PretrainedConfig) and can be used to control the model outputs. Read the documentation from [PretrainedConfig](https://huggingface.co/docs/transformers/main/en/main_classes/configuration" \l "transformers.PretrainedConfig) for more information.

Copied

>>> from transformers import LlamaModel, LlamaConfig

>>> *# Initializing a LLaMA llama-7b style configuration*

>>> configuration = LlamaConfig()

>>> *# Initializing a model from the llama-7b style configuration*

>>> model = LlamaModel(configuration)

>>> *# Accessing the model configuration*

>>> configuration = model.config

**LlamaTokenizer**

classtransformers.**LlamaTokenizer**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama.py#L66)

( vocab\_fileunk\_token = '<unk>'bos\_token = '<s>'eos\_token = '</s>'pad\_token = Nonesp\_model\_kwargs: typing.Union[typing.Dict[str, typing.Any], NoneType] = Noneadd\_bos\_token = Trueadd\_eos\_token = Falseclean\_up\_tokenization\_spaces = Falseuse\_default\_system\_prompt = Falsespaces\_between\_special\_tokens = Falselegacy = None\*\*kwargs )

**Parameters**

* **vocab\_file** (str) — Path to the vocabulary file.
* **unk\_token** (str or tokenizers.AddedToken, *optional*, defaults to "<unk>") — The unknown token. A token that is not in the vocabulary cannot be converted to an ID and is set to be this token instead.
* **bos\_token** (str or tokenizers.AddedToken, *optional*, defaults to "<s>") — The beginning of sequence token that was used during pretraining. Can be used a sequence classifier token.
* **eos\_token** (str or tokenizers.AddedToken, *optional*, defaults to "</s>") — The end of sequence token.
* **pad\_token** (str or tokenizers.AddedToken, *optional*) — A special token used to make arrays of tokens the same size for batching purpose. Will then be ignored by attention mechanisms or loss computation.
* **sp\_model\_kwargs** (Dict[str, Any], Optional, *optional*) — Will be passed to the SentencePieceProcessor.\_\_init\_\_() method. The [Python wrapper for SentencePiece](https://github.com/google/sentencepiece/tree/master/python) can be used, among other things, to set:
  + enable\_sampling: Enable subword regularization.
  + nbest\_size: Sampling parameters for unigram. Invalid for BPE-Dropout.
    - nbest\_size = {0,1}: No sampling is performed.
    - nbest\_size > 1: samples from the nbest\_size results.
    - nbest\_size < 0: assuming that nbest\_size is infinite and samples from the all hypothesis (lattice) using forward-filtering-and-backward-sampling algorithm.
  + alpha: Smoothing parameter for unigram sampling, and dropout probability of merge operations for BPE-dropout.
* **add\_bos\_token** (bool, *optional*, defaults to True) — Whether or not to add an bos\_token at the start of sequences.
* **add\_eos\_token** (bool, *optional*, defaults to False) — Whether or not to add an eos\_token at the end of sequences.
* **clean\_up\_tokenization\_spaces** (bool, *optional*, defaults to False) — Whether or not to cleanup spaces after decoding, cleanup consists in removing potential artifacts like extra spaces.
* **use\_default\_system\_prompt** (bool, *optional*, defaults to False) — Whether or not the default system prompt for Llama should be used.
* **spaces\_between\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not to add spaces between special tokens.
* **legacy** (bool, *optional*) — Whether or not the legacy behavior of the tokenizer should be used. Legacy is before the merge of #24622 and #25224 which includes fixes to properly handle tokens that appear after special tokens. A simple example:
  + legacy=True:

Construct a Llama tokenizer. Based on byte-level Byte-Pair-Encoding. The default padding token is unset as there is no padding token in the original model.

build\_inputs\_with\_special\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama.py#L333)

( token\_ids\_0token\_ids\_1 = None )

get\_special\_tokens\_mask

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama.py#L344)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = Nonealready\_has\_special\_tokens: bool = False ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — List of IDs.
* **token\_ids\_1** (List[int], *optional*) — Optional second list of IDs for sequence pairs.
* **already\_has\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not the token list is already formatted with special tokens for the model.

**Returns**

List[int]

A list of integers in the range [0, 1]: 1 for a special token, 0 for a sequence token.

Retrieve sequence ids from a token list that has no special tokens added. This method is called when adding special tokens using the tokenizer prepare\_for\_model method.

create\_token\_type\_ids\_from\_sequences

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama.py#L381)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = None ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — List of ids.
* **token\_ids\_1** (List[int], *optional*) — Optional second list of IDs for sequence pairs.

**Returns**

List[int]

List of [token type IDs](https://huggingface.co/docs/transformers/main/glossary#token-type-ids) according to the given sequence(s).

Creates a mask from the two sequences passed to be used in a sequence-pair classification task. An ALBERT

sequence pair mask has the following format:

Copied

0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1

| first sequence | second sequence |

if token\_ids\_1 is None, only returns the first portion of the mask (0s).

save\_vocabulary

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama.py#L306)

( save\_directoryfilename\_prefix: typing.Optional[str] = None ) **→** Tuple(str)

**Parameters**

* **save\_directory** (str) — The directory in which to save the vocabulary.

**Returns**

Tuple(str)

Paths to the files saved.

Save the vocabulary and special tokens file to a directory.

**LlamaTokenizerFast**

classtransformers.**LlamaTokenizerFast**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama_fast.py#L57)

( vocab\_file = Nonetokenizer\_file = Noneclean\_up\_tokenization\_spaces = Falseunk\_token = '<unk>'bos\_token = '<s>'eos\_token = '</s>'add\_bos\_token = Trueadd\_eos\_token = Falseuse\_default\_system\_prompt = False\*\*kwargs )

**Parameters**

* **vocab\_file** (str, *optional*) — [SentencePiece](https://github.com/google/sentencepiece) file (generally has a .model extension) that contains the vocabulary necessary to instantiate a tokenizer.
* **tokenizer\_file** (str, *optional*) — [tokenizers](https://github.com/huggingface/tokenizers) file (generally has a .json extension) that contains everything needed to load the tokenizer.
* **clean\_up\_tokenization\_spaces** (bool, *optional*, defaults to False) — Whether or not to cleanup spaces after decoding, cleanup consists in removing potential artifacts like extra spaces.
* **unk\_token** (str or tokenizers.AddedToken, *optional*, defaults to "<unk>") — The unknown token. A token that is not in the vocabulary cannot be converted to an ID and is set to be this token instead.
* **bos\_token** (str or tokenizers.AddedToken, *optional*, defaults to "<s>") — The beginning of sequence token that was used during pretraining. Can be used a sequence classifier token.
* **eos\_token** (str or tokenizers.AddedToken, *optional*, defaults to "</s>") — The end of sequence token.
* **add\_bos\_token** (bool, *optional*, defaults to True) — Whether or not to add an bos\_token at the start of sequences.
* **add\_eos\_token** (bool, *optional*, defaults to False) — Whether or not to add an eos\_token at the end of sequences.
* **use\_default\_system\_prompt** (bool, *optional*, defaults to False) — Whether or not the default system prompt for Llama should be used.

Construct a Llama tokenizer. Based on byte-level Byte-Pair-Encoding.

This uses notably ByteFallback and no normalization.

Copied

>>> from transformers import LlamaTokenizerFast

>>> tokenizer = LlamaTokenizerFast.from\_pretrained("hf-internal-testing/llama-tokenizer")

>>> tokenizer.encode("Hello this is a test")

[1, 15043, 445, 338, 263, 1243]

If you want to change the bos\_token or the eos\_token, make sure to specify them when initializing the model, or call tokenizer.update\_post\_processor() to make sure that the post-processing is correctly done (otherwise the values of the first token and final token of an encoded sequence will not be correct). For more details, checkout [post-processors] (<https://huggingface.co/docs/tokenizers/api/post-processors>) documentation.

This tokenizer inherits from [PreTrainedTokenizerFast](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer" \l "transformers.PreTrainedTokenizerFast) which contains most of the main methods. Users should refer to this superclass for more information regarding those methods.

build\_inputs\_with\_special\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama_fast.py#L272)

( token\_ids\_0token\_ids\_1 = None )

get\_special\_tokens\_mask

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L3766)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = Nonealready\_has\_special\_tokens: bool = False ) **→** A list of integers in the range [0, 1]

**Parameters**

* **token\_ids\_0** (List[int]) — List of ids of the first sequence.
* **token\_ids\_1** (List[int], *optional*) — List of ids of the second sequence.
* **already\_has\_special\_tokens** (bool, *optional*, defaults to False) — Whether or not the token list is already formatted with special tokens for the model.

**Returns**

**A list of integers in the range [0, 1]**

1 for a special token, 0 for a sequence token.

Retrieves sequence ids from a token list that has no special tokens added. This method is called when adding special tokens using the tokenizer prepare\_for\_model or encode\_plus methods.

create\_token\_type\_ids\_from\_sequences

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils_base.py#L3301)

( token\_ids\_0: typing.List[int]token\_ids\_1: typing.Optional[typing.List[int]] = None ) **→** List[int]

**Parameters**

* **token\_ids\_0** (List[int]) — The first tokenized sequence.
* **token\_ids\_1** (List[int], *optional*) — The second tokenized sequence.

**Returns**

List[int]

The token type ids.

Create the token type IDs corresponding to the sequences passed. [What are token type IDs?](https://huggingface.co/docs/transformers/main/glossary#token-type-ids)

Should be overridden in a subclass if the model has a special way of building those.

update\_post\_processor

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama_fast.py#L146)

( )

Updates the underlying post processor with the current bos\_token and eos\_token.

save\_vocabulary

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/tokenization_llama_fast.py#L190)

( save\_directory: strfilename\_prefix: typing.Optional[str] = None )

**LlamaModel**

classtransformers.**LlamaModel**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py#L943)

( config: LlamaConfig )

**Parameters**

* **config** ([LlamaConfig](https://huggingface.co/docs/transformers/main/en/model_doc/llama" \l "transformers.LlamaConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model" \l "transformers.PreTrainedModel.from_pretrained) method to load the model weights. config — LlamaConfig

The bare LLaMA Model outputting raw hidden-states without any specific head on top. This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model" \l "transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html" \l "torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

Transformer decoder consisting of *config.num\_hidden\_layers* layers. Each layer is a LlamaDecoderLayer

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py#L974)

( input\_ids: LongTensor = Noneattention\_mask: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.LongTensor] = Nonepast\_key\_values: typing.Optional[typing.List[torch.FloatTensor]] = Noneinputs\_embeds: typing.Optional[torch.FloatTensor] = Noneuse\_cache: typing.Optional[bool] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None )

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary. Padding will be ignored by default should you provide it.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto" \l "transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto" \l "transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

If past\_key\_values is used, optionally only the last input\_ids have to be input (see past\_key\_values).

If you want to change padding behavior, you should read modeling\_opt.\_prepare\_decoder\_attention\_mask and modify to your needs. See diagram 1 in [the paper](https://arxiv.org/abs/1910.13461) for more information on the default strategy.

* + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.n\_positions - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **past\_key\_values** (Cache or tuple(tuple(torch.FloatTensor)), *optional*) — Pre-computed hidden-states (key and values in the self-attention blocks and in the cross-attention blocks) that can be used to speed up sequential decoding. This typically consists in the past\_key\_values returned by the model at a previous stage of decoding, when use\_cache=True or config.use\_cache=True.

Two formats are allowed:

* + a [Cache](https://huggingface.co/docs/transformers/main/en/internal/generation_utils#transformers.Cache) instance;
  + Tuple of tuple(torch.FloatTensor) of length config.n\_layers, with each tuple having 2 tensors of shape (batch\_size, num\_heads, sequence\_length, embed\_size\_per\_head)). This is also known as the legacy cache format.

The model will output the same cache format that is fed as input. If no past\_key\_values are passed, the legacy cache format will be returned.

If past\_key\_values are used, the user can optionally input only the last input\_ids (those that don’t have their past key value states given to this model) of shape (batch\_size, 1) instead of all input\_ids of shape (batch\_size, sequence\_length).

* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **use\_cache** (bool, *optional*) — If set to True, past\_key\_values key value states are returned and can be used to speed up decoding (see past\_key\_values).
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output" \l "transformers.utils.ModelOutput) instead of a plain tuple.

The [LlamaModel](https://huggingface.co/docs/transformers/main/en/model_doc/llama#transformers.LlamaModel) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

**LlamaForCausalLM**

classtransformers.**LlamaForCausalLM**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py#L1106)

( config )

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py#L1136)

( input\_ids: LongTensor = Noneattention\_mask: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.LongTensor] = Nonepast\_key\_values: typing.Optional[typing.List[torch.FloatTensor]] = Noneinputs\_embeds: typing.Optional[torch.FloatTensor] = Nonelabels: typing.Optional[torch.LongTensor] = Noneuse\_cache: typing.Optional[bool] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None ) **→** [transformers.modeling\_outputs.CausalLMOutputWithPast](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.CausalLMOutputWithPast) or tuple(torch.FloatTensor)

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary. Padding will be ignored by default should you provide it.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto" \l "transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto" \l "transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

If past\_key\_values is used, optionally only the last input\_ids have to be input (see past\_key\_values).

If you want to change padding behavior, you should read modeling\_opt.\_prepare\_decoder\_attention\_mask and modify to your needs. See diagram 1 in [the paper](https://arxiv.org/abs/1910.13461) for more information on the default strategy.

* + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.n\_positions - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **past\_key\_values** (Cache or tuple(tuple(torch.FloatTensor)), *optional*) — Pre-computed hidden-states (key and values in the self-attention blocks and in the cross-attention blocks) that can be used to speed up sequential decoding. This typically consists in the past\_key\_values returned by the model at a previous stage of decoding, when use\_cache=True or config.use\_cache=True.

Two formats are allowed:

* + a [Cache](https://huggingface.co/docs/transformers/main/en/internal/generation_utils#transformers.Cache) instance;
  + Tuple of tuple(torch.FloatTensor) of length config.n\_layers, with each tuple having 2 tensors of shape (batch\_size, num\_heads, sequence\_length, embed\_size\_per\_head)). This is also known as the legacy cache format.

The model will output the same cache format that is fed as input. If no past\_key\_values are passed, the legacy cache format will be returned.

If past\_key\_values are used, the user can optionally input only the last input\_ids (those that don’t have their past key value states given to this model) of shape (batch\_size, 1) instead of all input\_ids of shape (batch\_size, sequence\_length).

* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **use\_cache** (bool, *optional*) — If set to True, past\_key\_values key value states are returned and can be used to speed up decoding (see past\_key\_values).
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output" \l "transformers.utils.ModelOutput) instead of a plain tuple.

Args — labels (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*): Labels for computing the masked language modeling loss. Indices should either be in [0, ..., config.vocab\_size] or -100 (see input\_ids docstring). Tokens with indices set to -100 are ignored (masked), the loss is only computed for the tokens with labels in [0, ..., config.vocab\_size].

**Returns**

[transformers.modeling\_outputs.CausalLMOutputWithPast](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.CausalLMOutputWithPast)**or**tuple(torch.FloatTensor)

A [transformers.modeling\_outputs.CausalLMOutputWithPast](https://huggingface.co/docs/transformers/main/en/main_classes/output" \l "transformers.modeling_outputs.CausalLMOutputWithPast) or a tuple of torch.FloatTensor (if return\_dict=False is passed or when config.return\_dict=False) comprising various elements depending on the configuration ([LlamaConfig](https://huggingface.co/docs/transformers/main/en/model_doc/llama" \l "transformers.LlamaConfig)) and inputs.

* **loss** (torch.FloatTensor of shape (1,), *optional*, returned when labels is provided) — Language modeling loss (for next-token prediction).
* **logits** (torch.FloatTensor of shape (batch\_size, sequence\_length, config.vocab\_size)) — Prediction scores of the language modeling head (scores for each vocabulary token before SoftMax).
* **past\_key\_values** (tuple(tuple(torch.FloatTensor)), *optional*, returned when use\_cache=True is passed or when config.use\_cache=True) — Tuple of tuple(torch.FloatTensor) of length config.n\_layers, with each tuple having 2 tensors of shape (batch\_size, num\_heads, sequence\_length, embed\_size\_per\_head))

Contains pre-computed hidden-states (key and values in the self-attention blocks) that can be used (see past\_key\_values input) to speed up sequential decoding.

* **hidden\_states** (tuple(torch.FloatTensor), *optional*, returned when output\_hidden\_states=True is passed or when config.output\_hidden\_states=True) — Tuple of torch.FloatTensor (one for the output of the embeddings, if the model has an embedding layer, + one for the output of each layer) of shape (batch\_size, sequence\_length, hidden\_size).

Hidden-states of the model at the output of each layer plus the optional initial embedding outputs.

* **attentions** (tuple(torch.FloatTensor), *optional*, returned when output\_attentions=True is passed or when config.output\_attentions=True) — Tuple of torch.FloatTensor (one for each layer) of shape (batch\_size, num\_heads, sequence\_length, sequence\_length).

Attentions weights after the attention softmax, used to compute the weighted average in the self-attention heads.

The [LlamaForCausalLM](https://huggingface.co/docs/transformers/main/en/model_doc/llama" \l "transformers.LlamaForCausalLM) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.

Example:

Copied

>>> from transformers import AutoTokenizer, LlamaForCausalLM

>>> model = LlamaForCausalLM.from\_pretrained(PATH\_TO\_CONVERTED\_WEIGHTS)

>>> tokenizer = AutoTokenizer.from\_pretrained(PATH\_TO\_CONVERTED\_TOKENIZER)

>>> prompt = "Hey, are you conscious? Can you talk to me?"

>>> inputs = tokenizer(prompt, return\_tensors="pt")

>>> *# Generate*

>>> generate\_ids = model.generate(inputs.input\_ids, max\_length=30)

>>> tokenizer.batch\_decode(generate\_ids, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=False)[0]

"Hey, are you conscious? Can you talk to me?\nI'm not conscious, but I can talk to you."

**LlamaForSequenceClassification**

classtransformers.**LlamaForSequenceClassification**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py#L1310)

( config )

**Parameters**

* **config** ([LlamaConfig](https://huggingface.co/docs/transformers/main/en/model_doc/llama" \l "transformers.LlamaConfig)) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the [from\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/model" \l "transformers.PreTrainedModel.from_pretrained) method to load the model weights.

The LLaMa Model transformer with a sequence classification head on top (linear layer).

[LlamaForSequenceClassification](https://huggingface.co/docs/transformers/main/en/model_doc/llama#transformers.LlamaForSequenceClassification) uses the last token in order to do the classification, as other causal models (e.g. GPT-2) do.

Since it does classification on the last token, it requires to know the position of the last token. If a pad\_token\_id is defined in the configuration, it finds the last token that is not a padding token in each row. If no pad\_token\_id is defined, it simply takes the last value in each row of the batch. Since it cannot guess the padding tokens when inputs\_embeds are passed instead of input\_ids, it does the same (take the last value in each row of the batch).

This model inherits from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model" \l "transformers.PreTrainedModel). Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch [torch.nn.Module](https://pytorch.org/docs/stable/nn.html" \l "torch.nn.Module) subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

forward

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py#L1326)

( input\_ids: LongTensor = Noneattention\_mask: typing.Optional[torch.Tensor] = Noneposition\_ids: typing.Optional[torch.LongTensor] = Nonepast\_key\_values: typing.Optional[typing.List[torch.FloatTensor]] = Noneinputs\_embeds: typing.Optional[torch.FloatTensor] = Nonelabels: typing.Optional[torch.LongTensor] = Noneuse\_cache: typing.Optional[bool] = Noneoutput\_attentions: typing.Optional[bool] = Noneoutput\_hidden\_states: typing.Optional[bool] = Nonereturn\_dict: typing.Optional[bool] = None )

**Parameters**

* **input\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length)) — Indices of input sequence tokens in the vocabulary. Padding will be ignored by default should you provide it.

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto" \l "transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

[What are input IDs?](https://huggingface.co/docs/transformers/main/glossary#input-ids)

* **attention\_mask** (torch.Tensor of shape (batch\_size, sequence\_length), *optional*) — Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]:
  + 1 for tokens that are **not masked**,
  + 0 for tokens that are **masked**.

[What are attention masks?](https://huggingface.co/docs/transformers/main/glossary#attention-mask)

Indices can be obtained using [AutoTokenizer](https://huggingface.co/docs/transformers/main/en/model_doc/auto" \l "transformers.AutoTokenizer). See [PreTrainedTokenizer.encode()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.encode) and [PreTrainedTokenizer.**call**()](https://huggingface.co/docs/transformers/main/en/main_classes/tokenizer#transformers.PreTrainedTokenizerFast.__call__) for details.

If past\_key\_values is used, optionally only the last input\_ids have to be input (see past\_key\_values).

If you want to change padding behavior, you should read modeling\_opt.\_prepare\_decoder\_attention\_mask and modify to your needs. See diagram 1 in [the paper](https://arxiv.org/abs/1910.13461) for more information on the default strategy.

* + 1 indicates the head is **not masked**,
  + 0 indicates the head is **masked**.
* **position\_ids** (torch.LongTensor of shape (batch\_size, sequence\_length), *optional*) — Indices of positions of each input sequence tokens in the position embeddings. Selected in the range [0, config.n\_positions - 1].

[What are position IDs?](https://huggingface.co/docs/transformers/main/glossary#position-ids)

* **past\_key\_values** (Cache or tuple(tuple(torch.FloatTensor)), *optional*) — Pre-computed hidden-states (key and values in the self-attention blocks and in the cross-attention blocks) that can be used to speed up sequential decoding. This typically consists in the past\_key\_values returned by the model at a previous stage of decoding, when use\_cache=True or config.use\_cache=True.

Two formats are allowed:

* + a [Cache](https://huggingface.co/docs/transformers/main/en/internal/generation_utils#transformers.Cache) instance;
  + Tuple of tuple(torch.FloatTensor) of length config.n\_layers, with each tuple having 2 tensors of shape (batch\_size, num\_heads, sequence\_length, embed\_size\_per\_head)). This is also known as the legacy cache format.

The model will output the same cache format that is fed as input. If no past\_key\_values are passed, the legacy cache format will be returned.

If past\_key\_values are used, the user can optionally input only the last input\_ids (those that don’t have their past key value states given to this model) of shape (batch\_size, 1) instead of all input\_ids of shape (batch\_size, sequence\_length).

* **inputs\_embeds** (torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size), *optional*) — Optionally, instead of passing input\_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input\_ids indices into associated vectors than the model’s internal embedding lookup matrix.
* **use\_cache** (bool, *optional*) — If set to True, past\_key\_values key value states are returned and can be used to speed up decoding (see past\_key\_values).
* **output\_attentions** (bool, *optional*) — Whether or not to return the attentions tensors of all attention layers. See attentions under returned tensors for more detail.
* **output\_hidden\_states** (bool, *optional*) — Whether or not to return the hidden states of all layers. See hidden\_states under returned tensors for more detail.
* **return\_dict** (bool, *optional*) — Whether or not to return a [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output" \l "transformers.utils.ModelOutput) instead of a plain tuple.
* **labels** (torch.LongTensor of shape (batch\_size,), *optional*) — Labels for computing the sequence classification/regression loss. Indices should be in [0, ..., config.num\_labels - 1]. If config.num\_labels == 1 a regression loss is computed (Mean-Square loss), If config.num\_labels > 1 a classification loss is computed (Cross-Entropy).

The [LlamaForSequenceClassification](https://huggingface.co/docs/transformers/main/en/model_doc/llama" \l "transformers.LlamaForSequenceClassification) forward method, overrides the \_\_call\_\_ special method.

Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the pre and post processing steps while the latter silently ignores them.