Philosophy

🤗 Transformers is an opinionated library built for:

- NLP researchers and educators seeking to use/study/extend large-scale transformers models
- hands-on practitioners who want to fine-tune those models and/or serve them in production
- engineers who just want to download a pretrained model and use it to solve a given NLP task.

The library was designed with two strong goals in mind:

- Be as easy and fast to use as possible:
    - We strongly limited the number of user-facing abstractions to learn, in fact, there are almost no abstractions, just three standard classes required to use each model: configuration, models and tokenizer.
    - All of these classes can be initialized in a simple and unified way from pretrained instances by using a common `from_pretrained()` instantiation method which will take care of downloading (if needed), caching and loading the related class instance and associated data (configurations' hyper-parameters, tokenizers' vocabulary, and models' weights) from a pretrained checkpoint provided on Hugging Face Hub or your own saved checkpoint.
    - On top of those three base classes, the library provides two APIs: `pipeline()` for quickly using a model (plus its associated tokenizer and configuration) on a given task and `Trainer()` / `TFTrainer()` to quickly train or fine-tune a given model.
    - As a consequence, this library is NOT a modular toolbox of building blocks for neural nets. If you want to extend/build-upon the library, just use regular Python/PyTorch/TensorFlow/Keras modules and inherit from the base classes of the library to reuse functionalities like model loading/saving.

- Provide state-of-the-art models with performances as close as possible to the original models:
    - We provide at least one example for each architecture which reproduces a result provided by the official authors of said architecture.
    - The code is usually as close to the original code base as possible which means some PyTorch code may be not as *pytorchic* as it could be as a result of being converted TensorFlow code and vice versa.

A few other goals:

- Expose the models' internals as consistently as possible:
    - We give access, using a single API, to the full hidden-states and attention weights.
    - Tokenizer and base model's API are standardized to easily switch between models.
- Incorporate a subjective selection of promising tools for fine-tuning/investigating these models:
    - A simple/consistent way to add new tokens to the vocabulary and embeddings for fine-tuning.
    - Simple ways to mask and prune transformer heads.
- Switch easily between PyTorch and TensorFlow 2.0, allowing training using one framwork and inference using another.

Main concepts

The library is build around three types of classes for each model:

- **Model classes** such as `BertModel`, which are 30+ PyTorch models (torch.nn.Module) or Keras models (tf.keras.Model) that work with the pretrained weights provided in the library.
- **Configuration classes** such as `BertConfig`, which store all the parameters required to build a model. You don't always need to instantiate these yourself. In particular, if you are using a pretrained model without any modification, creating the model will automatically take care of instantiating the configuration (which is part of the model).
- **Tokenizer classes** such as `BertTokenizer`, which store the vocabulary for each model and provide methods for encoding/decoding strings in a list of token embeddings indices to be fed to a model.

All these classes can be instantiated from pretrained instances and saved locally using two methods:

- `from_pretrained()` let you instantiate a model/configuration/tokenizer from a pretrained version either provided by the library itself (the suported models are provided in the list here or stored locally (or on a server) by the user,
- `save_pretrained()` let you save a model/configuration/tokenizer locally so that it can be reloaded using `from_pretrained()`.