# Transformers: State-of-the-Art Natural Language Processing

**Conference Paper** · January 2020

DOI: 10.18653/v1/2020.emnlp-demos.6



**22 authors**, including:

Joe Davison
Hugging Face
**4** PUBLICATIONS **598** CITATIONS

SEE PROFILE

# Transformers: State-of-the-Art Natural Language Processing

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, Alexander M. Rush

Hugging Face, Brooklyn, USA / {first-name}@huggingface.co

## Abstract

Recent progress in natural language processing has been driven by advances in both model architecture and model pretraining. Transformer architectures have facilitated building higher-capacity models and pretraining has made it possible to effectively utilize this capacity for a wide variety of tasks. *Transformers* is an open-source library with the goal of opening up these advances to the wider machine learning community. The library consists of carefully engineered state-of-the art Transformer architectures under a unified API. Backing this library is a curated collection of pretrained models made by and available for the community. *Transformers* is designed to be extensible by researchers, simple for practitioners, and fast and robust in industrial deployments. The library is available at https://github.com/huggingface/transformers.

## 1 Introduction

The Transformer (Vaswani et al., 2017) has rapidly become the dominant architecture for natural language processing, surpassing alternative neural models such as convolutional and recurrent neural networks in performance for tasks in both natural language understanding and natural language generation. The architecture scales with training data and model size, facilitates efficient parallel training, and captures long-range sequence features.

Model pretraining (McCann et al., 2017; Howard and Ruder, 2018; Peters et al., 2018; Devlin et al., 2018) allows models to be trained on generic corpora and subsequently be easily adapted to specific tasks with strong performance. The Transformer architecture is particularly conducive to pretraining on large text corpora, leading to major gains in accuracy on downstream tasks including text classification (Yang et al., 2019), language understanding (Liu et al., 2019b; Wang et al., 2018, 2019), machine translation (Lample and Conneau, 2019a), coreference resolution (Joshi et al., 2019), commonsense inference (Bosselut et al., 2019), and summarization (Lewis et al., 2019) among others.

This advance leads to a wide range of practical challenges that must be addressed in order for these models to be widely utilized. The ubiquitous use of the Transformer calls for systems to train, analyze, scale, and augment the model on a variety of platforms. The architecture is used as a building block to design increasingly sophisticated extensions and precise experiments. The pervasive adoption of pretraining methods has led to the need to distribute, fine-tune, deploy, and compress the core pretrained models used by the community.

*Transformers* is a library dedicated to supporting Transformer-based architectures and facilitating the distribution of pretrained models. At the core of the libary is an implementation of the Transformer which is designed for both research and production. The philosophy is to support industrial-strength implementations of popular model variants that are easy to read, extend, and deploy. On this foundation, the library supports the distribution and usage of a wide-variety of pretrained models in a centralized model hub. This hub supports users to compare different models with the same minimal API and to experiment with shared models on a variety of different tasks.

*Transformers* is an ongoing effort maintained by the team of engineers and researchers at Hugging Face with support from a vibrant community of over 400 external contributors. The library is released under the Apache 2.0 license and is available on GitHub[1]. Detailed documentation and tutorials are available on Hugging Face's website[2].

[1] https://github.com/huggingface/transformers

[2] https://huggingface.co/transformers/



*Proceedings of the 2020 EMNLP (Systems Demonstrations)*, pages 38–45

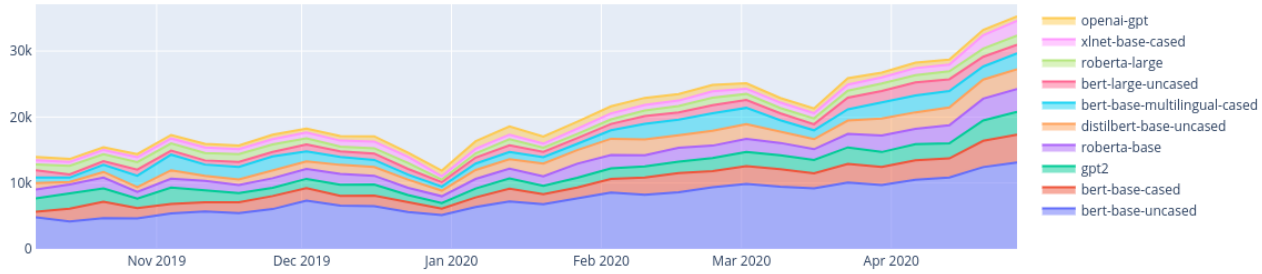



Figure 1: Average daily unique downloads of the most downloaded pretrained models, Oct. 2019 to May 2020.

## 2 Related Work

The NLP and ML communities have a strong culture of building open-source research tools. The structure of *Transformers* is inspired by the pioneering tensor2tensor library (Vaswani et al., 2018) and the original source code for BERT (Devlin et al., 2018), both from Google Research. The concept of providing easy caching for pretrained models stemmed from AllenNLP (Gardner et al., 2018). The library is also closely related to neural translation and language modeling systems, such as Fairseq (Ott et al., 2019), OpenNMT (Klein et al., 2017), Texar (Hu et al., 2018), Megatron-LM (Shoeybi et al., 2019), and Marian NMT (Junczys-Dowmunt et al., 2018). Building on these elements, *Transformers* adds extra user-facing features to allow for easy downloading, caching, and fine-tuning of the models as well as seamless transition to production. *Transformers* maintains some compatibility with these libraries, most directly including a tool for performing inference using models from Marian NMT and Google's BERT.

There is a long history of easy-to-use, user-facing libraries for general-purpose NLP. Two core libraries are NLTK (Loper and Bird, 2002) and Stanford CoreNLP (Manning et al., 2014), which collect a variety of different approaches to NLP in a single package. More recently, general-purpose, open-source libraries have focused primarily on machine learning for a variety of NLP tasks, these include Spacy (Honnibal and Montani, 2017), AllenNLP (Gardner et al., 2018), flair (Akbik et al., 2019), and Stanza (Qi et al., 2020). *Transformers* provides similar functionality as these libraries. Additionally, each of these libraries now uses the *Transformers* library and model hub as a low-level framework.

Since *Transformers* provides a hub for NLP models, it is also related to popular model hubs including Torch Hub and TensorFlow Hub which collect framework-specific model parameters for easy use. Unlike these hubs, *Transformers* is domain-specific which allows the system to provide automatic support for model analysis, usage, deployment, benchmarking, and easy replicability.

## 3 Library Design

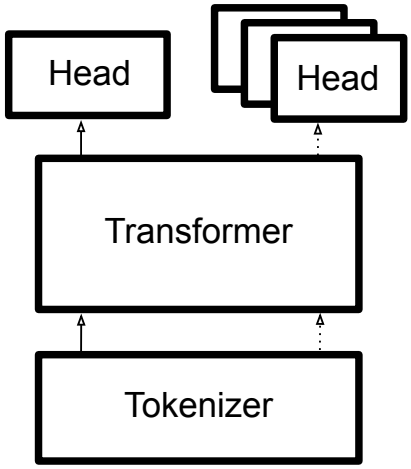
*Transformers* is designed to mirror the standard NLP machine learning model pipeline: process data, apply a model, and make predictions. Although the library includes tools facilitating training and development, in this technical report we focus on the core modeling specifications. For complete details about the features of the library refer to the documentation available on `https://huggingface.co/transformers/`.

Every model in the library is fully defined by three building blocks shown in the diagram in Figure 2: (a) a tokenizer, which converts raw text to sparse index encodings, (b) a transformer, which transforms sparse indices to contextual embeddings, and (c) a head, which uses contextual embeddings to make a task-specific prediction. Most user needs can be addressed with these three components.

**Transformers** Central to the library are carefully tested implementations of Transformer architecture variants which are widely used in NLP. The full list of currently implemented architectures is shown in Figure 2 (Left). While each of these architectures

| Heads | | | | |
|---|---|---|---|---|
| Name | Input | Output | Tasks | Ex. Datasets |
| Language Modeling | $x_{1:n-1}$ | $x_n \in \mathcal{V}$ | Generation | WikiText-103 |
| Sequence Classification | $x_{1:N}$ | $y \in \mathcal{C}$ | Classification, Sentiment Analysis | GLUE, SST, MNLI |
| Question Answering | $x_{1:M}$,$x_{M:N}$ | $y$ span $[1:N]$ | QA, Reading Comprehension | SQuAD, Natural Questions |
| Token Classification | $x_{1:N}$ | $y_{1:N} \in \mathcal{C}^N$ | NER, Tagging | OntoNotes, WNUT |
| Multiple Choice | $x_{1:N}, \mathcal{X}$ | $y \in \mathcal{X}$ | Text Selection | SWAG, ARC |
| Masked LM | $x_{1:N \setminus n}$ | $x_n \in \mathcal{V}$ | Pretraining | Wikitext, C4 |
| Conditional Generation | $x_{1:N}$ | $y_{1:M} \in \mathcal{V}^M$ | Translation, Summarization | WMT, IWSLT, CNN/DM, XSum |

| Transformers | |
|---|---|
| Masked $[x_{1:N \setminus n} \Rightarrow x_n]$ | |
| BERT | (Devlin et al., 2018) |
| RoBERTa | (Liu et al., 2019a) |
| Autoregressive $[x_{1:n-1} \Rightarrow x_n]$ | |
| GPT / GPT-2 | (Radford et al., 2019) |
| Trans-XL | (Dai et al., 2019) |
| XLNet | (Yang et al., 2019) |
| Seq-to-Seq $[\sim x_{1:N} \Rightarrow x_{1:N}]$ | |
| BART | (Lewis et al., 2019) |
| T5 | (Raffel et al., 2019) |
| MarianMT | (J.-Dowmunt et al., 2018) |
| Specialty: Multimodal | |
| MMBT | (Kiela et al., 2019) |
| Specialty: Long-Distance | |
| Reformer | (Kitaev et al., 2020) |
| Longformer | (Beltagy et al., 2020) |
| Specialty: Efficient | |
| ALBERT | (Lan et al., 2019) |
| Electra | (Clark et al., 2020) |
| DistilBERT | (Sanh et al., 2019) |
| Specialty: Multilingual | |
| XLM/RoBERTa | (Lample and Conneau, 2019b) |




| Tokenizers | |
|---|---|
| Name | Ex. Uses |
| Character-Level BPE | NMT, GPT |
| Byte-Level BPE | GPT-2 |
| WordPiece | BERT |
| SentencePiece | XLNet |
| Unigram | LM |
| Character | Reformer |
| Custom | Bio-Chem |

Figure 2: The *Transformers* library. **(Diagram-Right)** Each model is made up of a Tokenizer, Transformer, and Head. The model is pretrained with a fixed head and can then be further fine-tuned with alternate heads for different tasks. **(Bottom)** Each model uses a specific Tokenizer either implemented in Python or in Rust. These often differ in small details, but need to be in sync with pretraining. **(Left)** Transformer architectures specialized for different tasks, e.g. understanding versus generation, or for specific use-cases, e.g. speed, image+text. **(Top)** heads allow a Transformer to be used for different tasks. Here we assume the input token sequence is $x_{1:N}$ from a vocabulary $\mathcal{V}$, and $y$ represents different possible outputs, possibly from a class set $\mathcal{C}$. Example datasets represent a small subset of example code distributed with the library.

shares the same multi-headed attention core, there are significant differences between them including positional representations, masking, padding, and the use of sequence-to-sequence design. Additionally, various models are built to target different applications of NLP such as understanding, generation, and conditional generation, plus specialized use cases such as fast inference or multi-lingual applications.

Practically, all models follow the same hierarchy of abstraction: a base class implements the model's computation graph from an encoding (projection on the embedding matrix) through the series of self-attention layers to the final encoder hidden states. The base class is specific to each model and closely follows the model's original implementation which gives users the flexibility to easily dissect the inner workings of each individual architecture. In most cases, each model is implemented in a single file to enable ease of extensibility.

Wherever possible, different architectures follow the same API allowing users to switch easily between different models. A set of `Auto` classes provides a unified API that enables very fast switching between models and even between frameworks. These classes automatically instantiate with the configuration specified by the user-specified pretrained model.

**Tokenizers** A critical NLP-specific aspect of the library is the implementations of the tokenizers necessary to use each model. Tokenizer classes (each inheriting from a common base class) can either be instantiated from a corresponding pretrained model or can be configured manually. These classes store the vocabulary token-to-index map for their corresponding model and handle the encoding and decoding of input sequences according to a model's specific tokenization process. The tokenizers implemented are shown in Figure 2 (Right). Users can easily modify tokenizer with interfaces to add additional token mappings, special tokens (such as classification or separation tokens), or otherwise resize the vocabulary.

Tokenizers can also implement additional useful features for the users. These range from token type indices in the case of sequence classification to maximum length sequence truncating taking into account the added model-specific special tokens (most pretrained Transformer models have a maximum sequence length).

For training on very large datasets, Python-based tokenization is often undesirably slow. In the most recent release, *Transformers* switched its implementation to use a highly-optimized tokenization library by default. This low-level library, available at `https://github.com/huggingface/tokenizers`, is written in Rust to speed up the tokenization procedure both during training and deployment.

**Heads** Each Transformer can be paired with one out of several ready-implemented heads with outputs amenable to common types of tasks. These heads are implemented as additional wrapper classes on top of the base class, adding a specific output layer, and optional loss function, on top of the Transformer's contextual embeddings. The full set of implemented heads are shown in Figure 2 (Top). These classes follow a similar naming pattern: `XXXForSequenceClassification` where `XXX` is the name of the model and can be used for adaptation (fine-tuning) or pretraining. Some heads, such as conditional generation, support extra functionality like sampling and beam search.

For pretrained models, we release the heads used to pretrain the model itself. For instance, for BERT we release the language modeling and next sentence prediction heads which allows easy for adaptation using the pretraining objectives. We also make it easy for users to utilize the same core Transformer parameters with a variety of other heads for finetuning. While each head can be used generally, the library also includes a collection of examples that show each head on real problems. These examples demonstrate how a pretrained model can be adapted with a given head to achieve state-of-the-art results on a large variety of NLP tasks.

## 4 Community Model Hub

*Transformers* aims to facilitate easy use and distribution of pretrained models. Inherently this is a community process; a single pretraining run facilitates fine-tuning on many specific tasks. The Model Hub makes it simple for any end-user to access a model for use with their own data. This hub now contains 2,097 user models, both pretrained and fine-tuned, from across the community. Figure 1 shows the increase and distribution of popular transformers over time. While core models like BERT and GPT-2 continue to be popular, other specialized models including DistilBERT (Sanh et al., 2019), which was developed for the library, are

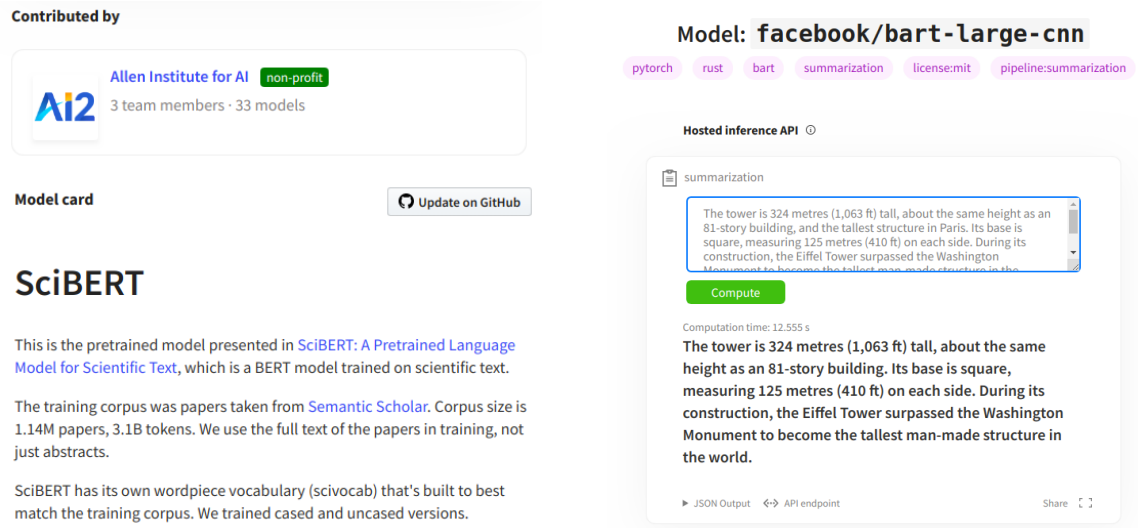



Figure 3: *Transformers* Model Hub. **(Left)** Example of a model page and model card for SciBERT (Beltagy et al., 2019), a pretrained model targeting extraction from scientific literature submitted by a community contributor. **(Right)** Example of an automatic inference widget for the pretrained BART (Lewis et al., 2019) model for summarization. Users can enter arbitrary text and a full version of the model is deployed on the fly to produce a summary.

now widely downloaded by the community.

The user interface of the Model Hub is designed to be simple and open to the community. To upload a model, any user can sign up for an account and use a command-line interface to produce an archive consisting a tokenizer, transformer, and head. This bundle may be a model trained through the library or converted from a checkpoint of other popular training tools. These models are then stored and given a canonical name which a user can use to download, cache, and run the model either for fine-tuning or inference in two lines of code. To load FlauBERT (Le et al., 2020), a BERT model pretrained on a French training corpus, the command is:

```
tknzr = AutoTokenizer.from_pretrained(
    "flaubert/flaubert_base_uncased")
model = AutoModel.from_pretrained(
    "flaubert/flaubert_base_uncased")
```

When a model is uploaded to the Model Hub, it is automatically given a landing page describing its core properties, architecture, and use cases. Additional model-specific metadata can be provided via a model card (Mitchell et al., 2018) that describes properties of its training, a citation to the work, datasets used during pretraining, and any caveats about known biases in the model and its predictions. An example model card is shown in Figure 3 (Left).

Since the Model Hub is specific to transformer-based models, we can target use cases that would be difficult for more general model collections. For example, because each uploaded model includes metadata concerning its structure, the model page can include live inference that allows users to experiment with output of models on a real data. Figure 3 (Right) shows an example of the model page with live inference. Additionally, model pages include links to other model-specific tools like benchmarking and visualizations. For example, model pages can link to exBERT (Hoover et al., 2019), a Transformer visualization library.

**Community Case Studies** The Model Hub highlights how *Transformers* is used by a variety of different community stakeholders. We summarize three specific observed use-cases in practice. We highlight specific systems developed by users with different goals following the architect, trainer, and end-user distinction of Strobelt et al. (2017):

*Case 1: Model Architects* AllenAI, a major NLP research lab, developed a new pretrained model for improved extraction from biomedical texts called SciBERT (Beltagy et al., 2019). They were able to train the model utilizing data from PubMed to produce a masked language model with state-of-the-art results on targeted text. They then used the Model Hub to distribute the model and promote it as part of their CORD - COVID-19 challenge, making it trivial for the community to use.

*Case 2: Task Trainers* Researchers at NYU were

interested in developing a test bed for the performance of *Transformers* on a variety of different semantic recognition tasks. Their framework Jiant (Pruksachatkun et al., 2020) allows them to experiment with different ways of pretraining models and comparing their outputs. They used the *Transformers* API as a generic front-end and performed fine-tuning on a variety of different models, leading to research on the structure of BERT (Tenney et al., 2019).

*Case 3: Application Users* Plot.ly, a company focused on user dashboards and analytics, was interested in deploying a model for automatic document summarization. They wanted an approach that scaled well and was simple to deploy, but had no need to train or fine-tune the model. They were able to search the Model Hub and find *DistilBART*, a pretrained and fine-tuned summarization model designed for accurate, fast inference. They were able to run and deploy the model directly from the hub with no required research or ML expertise.

## 5 Deployment

An increasingly important goal of *Transformers* is to make it easy to efficiently deploy model to production. Different users have different production needs, and deployment often requires solving significantly different challenges than training. The library therefore allows for several different strategies for production deployment.

One core propery of the libary is that models are available both in PyTorch and TensorFlow, and there is interoperability between both frameworks. A model trained in one of frameworks can be saved through standard serialization and be reloaded from the saved files in the other framework seamlessly. This makes it particularly easy to switch from one framework to the other one along the model lifetime (training, serving, etc.).

Each framework has deployment recommendations. For example, in PyTorch, models are compatible with TorchScript, an intermediate representation of a PyTorch model that can then be run either in Python in a more efficient way, or in a high-performance environment such as C++. Fine-tuned models can thus be exported to production-friendly environment, and run through TorchServing. TensorFlow includes several serving options within its ecosystem, and these can be used directly.

*Transformers* can also export models to intermediate neural network formats for further compilation. It supports converting models to the Open Neural Network Exchange format (ONNX) for deployment. Not only does this allow the model to be run in a standardized interoperable format, but also leads to significant speed-ups. Figure 4 shows experiments run in collaboration with the ONNX team to optimize BERT, RoBERTa, and GPT-2 from the *Transformers* library. Using this intermediate format, ONNX was able to achieve nearly a 4x speedup on this model. The team is also experimenting with other promising intermediate formats such as JAX/XLA (Bradbury et al., 2018) and TVM (Chen et al., 2018).

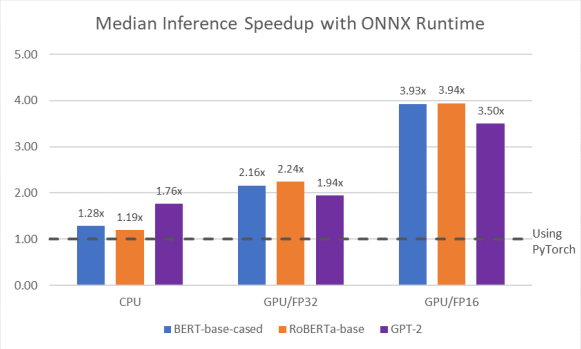



Figure 4: Experiments with *Transformers* inference in collaboration with ONNX.

Finally, as Transformers become more widely used in all NLP applications, it is increasingly important to deploy to edge devices such as phones or home electronics. Models can use adapters to convert models to *CoreML* weights that are suitable to be embedded inside a iOS application, to enable on-the-edge machine learning. Code is also made available[3]. Similar methods can be used for Android devices.

## 6 Conclusion

As Transformer and pretraining play larger roles in NLP, it is important for these models to be accessible to researchers and end-users. *Transformers* is an open-source library and community designed to facilitate users to access large-scale pretrained models, to build and experiment on top of them, and to deploy them in downstream tasks with state-of-the-art performance. *Transformers* has gained significant organic traction since its release and is set up to continue to provide core infrastructure while helping to facilitate access to new models.

[3] https://github.com/huggingface/swift-coreml-transformers

## References


a. PyTorch Hub. https://pytorch.org/hub/. Accessed: 2020-6-29.

b. TensorFlow hub. https://www.tensorflow.org/hub. Accessed: 2020-6-29.

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. Flair: An easy-to-use framework for state-of-the-art nlp. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59. aclweb.org.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The Long-Document transformer.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Çelikyilmaz, and Yejin Choi. 2019. Comet: Commonsense transformers for automatic knowledge graph construction. In *ACL*.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. 2018. JAX: composable transformations of Python+NumPy programs.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 578–594.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a Fixed-Length context.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 7(1).

Benjamin Hoover, Hendrik Strobelt, and Sebastian Gehrmann. 2019. exBERT: A visual analysis tool to explore learned representations in transformers models.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *ACL*.

Zhiting Hu, Haoran Shi, Bowen Tan, Wentao Wang, Zichao Yang, Tiancheng Zhao, Junxian He, Lianhui Qin, Di Wang, Xuezhe Ma, Zhengzhong Liu, Xiaodan Liang, Wangrong Zhu, Devendra Singh Sachan, and Eric P Xing. 2018. Texar: A modularized, versatile, and extensible toolkit for text generation.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F T Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in c++.

Douwe Kiela, Suvrat Bhooshan, Hamed Firooz, and Davide Testuggine. 2019. Supervised multimodal bitransformers for classifying images and text.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

Guillaume Lample and Alexis Conneau. 2019a. Cross-lingual language model pretraining. In *NeurIPS*.

Guillaume Lample and Alexis Conneau. 2019b. Cross-lingual language model pretraining.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations.

Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoît Crabbé, Laurent Besacier, and Didier Schwab. 2020. Flaubert: Unsupervised language model pre-training for french. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 2479–2490, Marseille, France. European Language Resources Association.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence pre-training for natural language generation, translation, and comprehension.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019a. RoBERTa: A robustly optimized BERT pretraining approach.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar S. Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Edward Loper and Steven Bird. 2002. NLTK: The natural language toolkit.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60. aclweb.org.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6294–6305. Curran Associates, Inc.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2018. Model cards for model reporting.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations.

Yada Pruksachatkun, Phil Yeres, Haokun Liu, Jason Phang, Phu Mon Htut, Alex Wang, Ian Tenney, and Samuel R Bowman. 2020. jiant: A software toolkit for research on general-purpose text understanding models. *arXiv preprint arXiv:2003.02249*.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D Manning. 2020. Stanza: A python natural language processing toolkit for many human languages.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified Text-to-Text transformer.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*.

Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. 2017. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. In *ACL*.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł Ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *ArXiv*, abs/1905.00537.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *ArXiv*, abs/1906.08237.