

Project Technical Report

| Project GitHub [link](#)

Topic

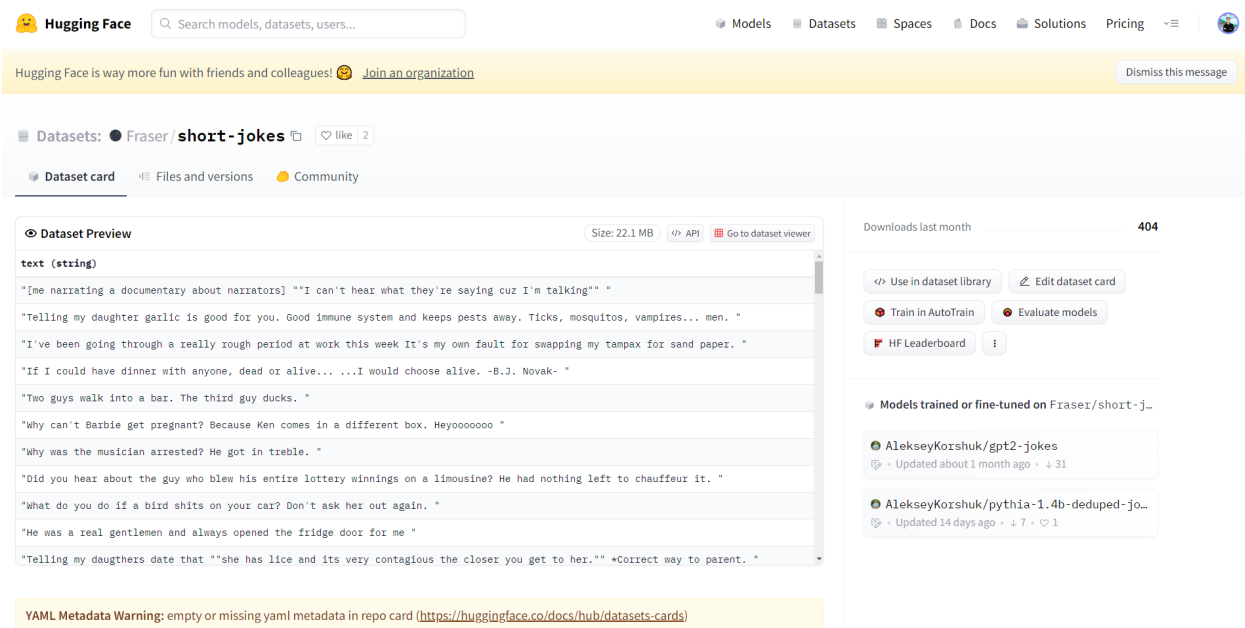
Jokes Generation (Decoder-based transformer)

Short description

The Joke Generation (Decoder-based transformer) is a natural language processing tool as a part of an NLP course at the university. The bot generates hilarious and witty jokes from a random seed, making use of natural language processing techniques to analyze the semantics of the input and produce humorous content. The motivation behind creating this bot is to demonstrate the capabilities of natural language processing while providing an amusing and engaging experience for users. With its ability to produce unique and diverse jokes, the Joke Generation is a prime example of how AI can be used to enhance our daily lives in ways that were previously unimaginable.

Data collection and processing

Dataset



The screenshot shows the Hugging Face website interface. At the top, there's a navigation bar with the Hugging Face logo, a search bar, and links for Models, Datasets, Spaces, Docs, Solutions, Pricing, and a user profile. Below the navigation bar, a yellow banner reads "Hugging Face is way more fun with friends and colleagues! Join an organization". The main content area displays the "Fraser/short-jokes" dataset card. The card has tabs for "Dataset card", "Files and versions", and "Community". The "Dataset card" tab is active, showing a "Dataset Preview" section with a list of text samples. To the right of the preview, there are buttons for "Use in dataset library", "Edit dataset card", "Train in AutoTrain", "Evaluate models", and "HF Leaderboard". Below these buttons, a section titled "Models trained or fine-tuned on Fraser/short-j_" lists two models: "AlekseyKorshuk/gpt2-jokes" and "AlekseyKorshuk/pythia-1.4b-deduped-jo...". At the bottom of the card, a yellow warning box states "YAML Metadata Warning: empty or missing yaml metadata in repo card (https://huggingface.co/docs/hub/datasets-cards)".

To test our hypotheses, we decided to take a shortened version of the huge dataset as **Fraser/short-jokes** dataset

The **Fraser/short-jokes** dataset available on the Hugging Face website is a collection of short, humorous jokes in the English language. This dataset contains over 200,000 jokes, which are categorized into different categories such as puns, one-liners, knock-knock jokes, and more.

Each joke in the dataset is represented as a text string and is associated with its corresponding category. The dataset can be used to train and test machine learning models for various natural language processing tasks such as joke generation, humor detection, and sentiment analysis.

This dataset is particularly useful for developing models that can generate humorous content, as it provides a large amount of diverse and high-quality examples of short jokes. Additionally, the dataset can be used for other applications such as building chatbots or language-based recommendation systems that incorporate humor into their responses.

Processing

```
def tokenize_function_no_pad(examples):  
    with CaptureLogger(tok_logger) as cl:  
        texts = examples[text_column_name]  
        texts = [text + "<|endoftext|>" for text in texts]  
        output = tokenizer(texts, padding=False, max_length=block_size, truncation=True)  
    return output
```

The function first captures the logging messages generated by a tokenizer logger, which could be used to monitor the tokenization process.

Then, the function extracts the text data from the `examples` input using a specific column name `text_column_name`, adds an `eos_token` (end-of-sequence token) to the end of each text, and stores them in a list called `texts`.

Next, the function tokenizes the `texts` using a tokenizer object, which is not shown in the code snippet. The `padding` argument is set to `False`, indicating that the tokenizer will not add any padding to the tokenized sequences. The `max_length` argument specifies the maximum length of the tokenized sequences, and the `truncation` argument indicates that any input text longer than `max_length` will be truncated to fit.

Finally, the function returns the output of the tokenization process. The output is a dictionary containing the tokenized input sequences, with keys such as `input_ids`, `attention_mask`, and `token_type_ids`, which are standard outputs of the Hugging Face transformers library. The `padding` tokens are not added in this function, so the tokenized sequences may have varying lengths.

Review of methods and models

We decide to use an **EleutherAI/pythia-1.4b-deduped** model

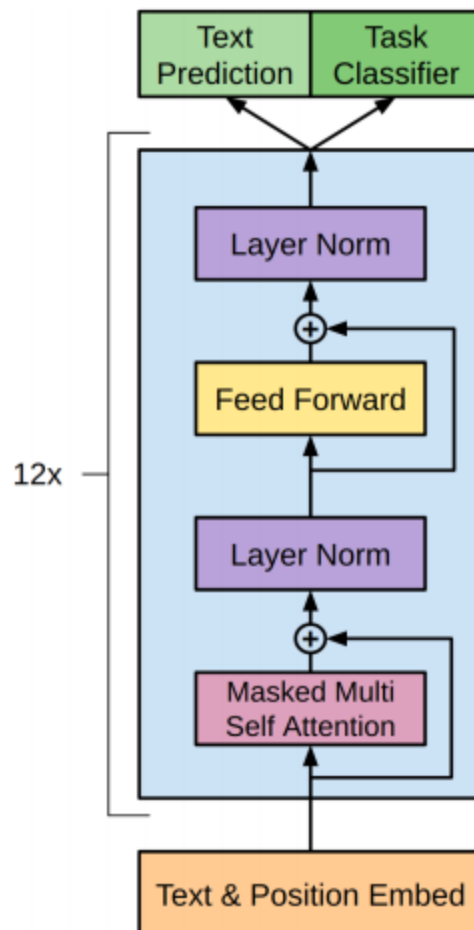
The screenshot shows the Hugging Face interface for the model **EleutherAI/pythia-1.4b-deduped**. The header includes the Hugging Face logo, a search bar, and navigation links for Models, Datasets, Spaces, Docs, Solutions, and Pricing. A banner message encourages joining an organization. The model card itself features a title bar with the model name, a 'like' button, and a list of tags including Text Generation, PyTorch, Transformers, and various languages. Below the title bar, there are tabs for Model card, Files and versions, and Community. The main content area contains a detailed description of the Pythia Scaling Suite, highlighting its purpose for interpretability research and its training on deduplicated data. A 'Details on previous early release and naming convention' link is provided. On the right side, there is a 'Hosted inference API' section with a 'Text Generation' dropdown, a text input field containing 'Once upon a time,', a 'Compute' button, and a 'JSON Output' checkbox. A line graph shows the download trend for the model, with 4,848 downloads in the last month.

The **Pythia Scaling Suite** is a collection of models developed to facilitate interpretability research. Trained on the Pile after the dataset has been globally deduplicated. Such model match or exceed the performance of similar and same-sized models, such as those in the OPT and GPT-Neo suites.

Paper: **Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling**

Architecture and implementation

Decoder-Only Architecture used



In this architecture, the model consists of only the decoder component of a sequence-to-sequence (seq2seq) model. It takes an input sequence and generates an output sequence without the encoder component. The decoder is usually an autoregressive model that generates each output token based on the previous tokens it has generated.

One of the advantages of using a Decoder-Only Architecture is that it is computationally more efficient than a full seq2seq model, as it requires fewer parameters to train. However, it may not perform as well as a full seq2seq model in tasks that require understanding the input sequence in depth, such as question answering or text classification.

We used distributed setup of **4xA100 with DeepSpeed**:

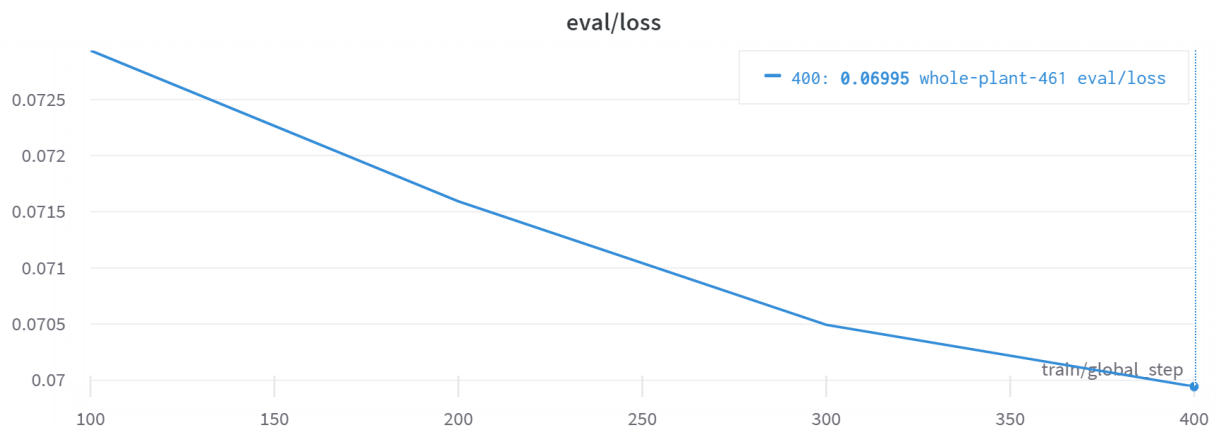
```
{
  "fp16": {
    "enabled": true,
    "loss_scale": 32,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "optimizer": {
    "type": "AdamW",
    "params": {
      "lr": "auto",
      "betas": "auto",
      "eps": "auto",
      "weight_decay": "auto"
    }
  },
  "scheduler": {
    "type": "WarmupLR",
    "params": {
      "warmup_min_lr": "auto",
      "warmup_max_lr": "auto",
      "warmup_num_steps": "auto"
    }
  },
  "zero_optimization": {
    "stage": 2,
    "overlap_comm": true,
    "contiguous_gradients": true,
    "sub_group_size": 1e9,
    "reduce_bucket_size": "auto",
    "stage3_prefetch_bucket_size": "auto",
    "stage3_param_persistence_threshold": "auto",
    "stage3_max_live_parameters": 1e9,
    "stage3_max_reuse_distance": 1e9,
    "stage3_gather_16bit_weights_on_model_save": true
  },
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "steps_per_print": 2000,
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "wall_clock_breakdown": false
}
```

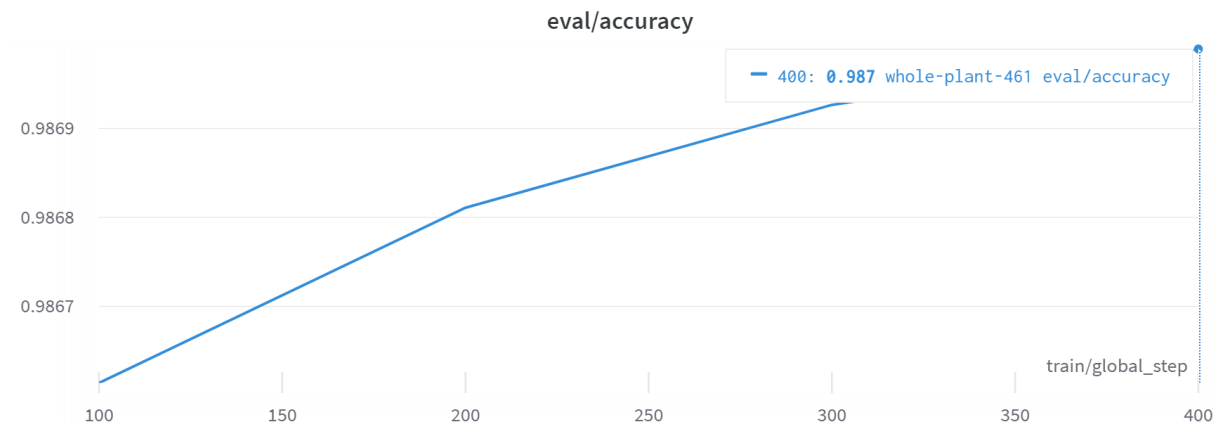
Hyperparameters

```
deepspeed run_clm.py \
  --model_name_or_path EleutherAI/pythia-1.4b-deduped \
  --dataset_name Fraser/short-jokes \
  --per_device_train_batch_size 32 \
  --per_device_eval_batch_size 32 \
  --gradient_accumulation_steps 1 \
  --do_train \
  --do_eval \
  --fp16 \
  --output_dir /tmp/test-clm \
  --report_to wandb \
  --num_train_epochs 1 \
  --push_to_hub \
  --hub_model_id AlekseyKorshuk/pythia-1.4b-deduped-jokes \
  --overwrite_output_dir \
  --evaluation_strategy "steps" \
  --eval_steps 100 \
  --save_strategy epoch \
  --save_steps 1 \
  --learning_rate 5e-5 \
  --warmup_ratio 0.05 \
  --deepspeed ds_config_soft.json
```

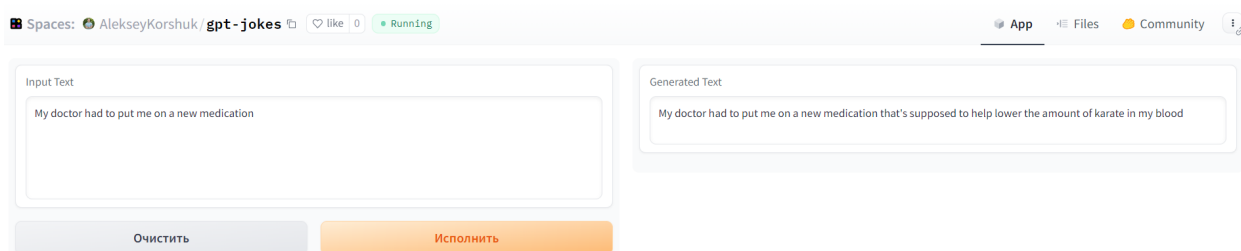
Evaluation and conclusion

Evaluation (wandb)





An example of the work of our model in the **playground**



Conclusion

The jock generation model is an AI-powered language model that has been trained on a dataset of jokes to generate new and original jokes. The idea behind the model is to create an automated way of generating humorous content that can be used for entertainment or marketing purposes. While the model is capable of generating jokes, the quality of the generated jokes is dependent on the quality of the trained data.

One of the main challenges with training a jock generation model is the need for high-quality training data. The model needs to be trained on a large dataset of jokes that are both funny and appropriate for the intended audience. If the dataset contains low-quality or inappropriate jokes, the model will likely generate similar jokes, which may not be suitable for the intended purpose.

Another challenge with the jock generation model is the need for human input to evaluate the generated jokes. While the model can generate jokes, it cannot evaluate

whether the joke is funny or not. Human input is required to determine whether the joke is humorous, appropriate, and relevant to the intended audience.

Despite these challenges, the joke generation model has shown promise in generating new and original jokes. However, it is important to note that the quality of the generated jokes is dependent on the quality of the trained data. Therefore, it is crucial to use high-quality training data to ensure that the model generates high-quality and appropriate jokes for the intended audience. Additionally, human input is necessary to evaluate the generated jokes to ensure that they meet the desired standards.

Each team member's contribution

- **Panov Evgenii** - search, analysis, and preparation of the dataset;
- **Rafail Venediktov** - search and refinement of the model;
- **Aliaksei Korshuk** - training model and making a playground;

Each member of the team has made an integral contribution to obtaining this result.

Resources

Fraser, A. (n.d.). Short Jokes Dataset. Hugging Face Datasets.

<https://huggingface.co/datasets/Fraser/short-jokes>

EleutherAI. (n.d.). Pythia-1.4b-deduped [Computer software]. Hugging Face. Retrieved March 27, 2023, from <https://huggingface.co/EleutherAI/pythia-1.4b-deduped>.

Korshuk, A. (n.d.). Hugging Face - d8dg1f9i. Weights & Biases.

<https://wandb.ai/aleksey-korshuk/huggingface/runs/d8dg1f9i?workspace=user->

Korshuk, A. (n.d.). GPT-Jokes [Computer software]. Hugging Face.

<https://huggingface.co/spaces/AlekseyKorshuk/gpt-jokes>.

Team GitHub Repository <https://github.com/rafailvv/nlp-joke-generation-bot>

DeppSpeed GitHub Repository <https://github.com/microsoft/DeepSpeed>