
Explorando o Universo das IAs com Hugging Face

Asimov Academy

ASIMOV

Conteúdo

| | |
|--|-----------|
| 01. O que é Hugging Face? | 5 |
| Afinal, o que é Hugging Face? | 5 |
| É tudo aberto mesmo? | 6 |
| Como usaremos o Hugging Face? | 6 |
| 02. A plataforma Hugging Face | 7 |
| Modelos | 7 |
| Página de um modelo | 8 |
| Datasets | 10 |
| Spaces | 10 |
| 03. Testando nossa primeira IA em tempo recorde | 14 |
| Nossa primeira IA | 14 |
| Dependências adicionais | 14 |
| Instalando o PyTorch | 15 |
| Rodando a IA pela primeira vez | 16 |
| Formatando a saída da IA | 16 |
| 04. Testando e comparando com outros modelos | 18 |
| Buscando por outros modelos | 18 |
| Como usar um modelo? | 19 |
| Consigo rodar o modelo? | 19 |
| Tamanho do arquivo pytorch_model.bin | 20 |
| Informação contida na etiqueta Safetensor | 22 |
| Cartão do modelo | 22 |
| E se não conseguir rodar no meu computador? | 23 |
| Testando os modelos escolhidos | 23 |
| 05. A biblioteca transformers | 25 |
| Tokenização: como modelos de IA enxergam o texto | 25 |
| A praticidade do pipeline | 26 |
| Acessando tokenizadores diretamente | 26 |
| Passando parâmetros para o pipeline | 27 |
| Quais tarefas podemos fazer? | 28 |

| | |
|---|-----------|
| 06. Modelos de conversação e geração de texto (chatbots) | 29 |
| Acessando o modelo | 29 |
| Ajustando o template da mensagem | 30 |
| Ajustando parâmetros do modelo | 30 |
| 07. Criando uma estrutura de chat | 31 |
| Considerações sobre a resposta do modelo | 32 |
| Formatando a saída do modelo | 32 |
| Criando um loop de conversa | 33 |
| O que aconteceu? | 35 |
| 08. Acessando modelos de IA através da Inference API | 36 |
| Conhecendo a Inference API | 36 |
| Acessando a Inference API | 36 |
| Entendendo a resposta do modelo | 37 |
| Templates de chat do Hugging Face | 38 |
| 09. Conversando com um chatbot pela Inference API | 39 |
| Adicionando o template correto | 39 |
| Expandindo para mais conversas | 39 |
| Loop de conversação pela Inference API | 40 |
| Porquê não usar a Inference API sempre? | 41 |
| 10. Modelos restritos e outras considerações | 43 |
| Modelos restritos | 43 |
| Autenticando com o token do Hugging Face | 44 |
| Criando a conta | 44 |
| Pedindo acesso a um modelo | 45 |
| Gerando um token de acesso | 46 |
| Adicionando o token no código | 46 |
| Melhores práticas ao usar o token: biblioteca dotenv | 47 |
| Como usar o arquivo .env | 47 |
| 11. Miniprojeto - webapp com múltiplos chatbots | 49 |
| 12. Utilizando IAs de tradução | 52 |
| IA de tradução | 52 |
| Parâmetros da tradução | 52 |
| Outros testes | 53 |
| Para quê usar modelos de tradução? | 55 |

| | |
|---|-----------|
| 13. Utilizando IAs de resumo | 56 |
| Resumindo textos em português | 56 |
| Resumindo um artigo da Wikipedia | 56 |
| Resumindo uma notícia | 58 |
| Outros parâmetros | 59 |
| 14. Classificando textos com IAs | 60 |
| Classificação de texto (análise de sentimento) | 60 |
| Classificando reviews de um produto | 60 |
| Outros exemplos de classificação | 61 |
| Classificando emoções | 61 |
| Classificando headlines de frases relacionadas a finanças | 62 |
| Classificando ironia em tweets | 62 |
| Classificação sem contexto (<i>zero-shot</i>) | 63 |
| Vale a pena usar um modelo <i>zero-shot</i> ? | 64 |
| 15. Acessando Datasets do Hugging Face | 65 |
| A anatomia de um Dataset | 65 |
| Acessando Datasets por Python | 66 |
| O que há no Dataset? | 67 |
| Splits | 67 |
| Linhas e colunas | 68 |
| Acessando dados | 68 |
| Convertendo um Dataset | 69 |
| Lidando com Datasets grandes | 69 |
| 16. Conhecendo a comunidade do Hugging Face | 71 |
| Página inicial | 71 |
| Opções de comunidade | 71 |
| Spaces | 72 |
| 17. Criando um Space com nosso webapp | 74 |
| Passo 1: criar um Space | 74 |
| Passo 2: adicionar seu código ao Space | 76 |
| Passo 3: Adicione as dependências | 78 |
| Passo 4: gerindo o token do Hugging Face | 82 |

| | |
|--|-----------|
| 18. Considerações sobre o deploy e encerramento | 86 |
| Configurações do Deploy | 86 |
| Hardware utilizado | 86 |
| Visibilidade do Space | 86 |
| Comunidade | 87 |
| Compartilhamento | 87 |
| Outras opções | 88 |
| Logs | 88 |
| Encerramento | 88 |

01. O que é Hugging Face?

Bem-vindos ao curso de Hugging Face da Asimov Academy!

Neste curso, vamos explorar as principais utilidades da plataforma de IA **Hugging Face**. Aprenderemos como utilizar a plataforma ao máximo, e como incorporar as bibliotecas de Python do Hugging Face aos nossos scripts de Python.

Afinal, o que é Hugging Face?

A Hugging Face é uma empresa que iniciou em 2017 na França, com o desenvolvimento de Chatbots. Com o tempo, passaram a desenvolver uma infraestrutura própria e bibliotecas de Python que simplificam o uso de modelos de NLP (processamento de linguagem natural, do inglês *natural language processing*).

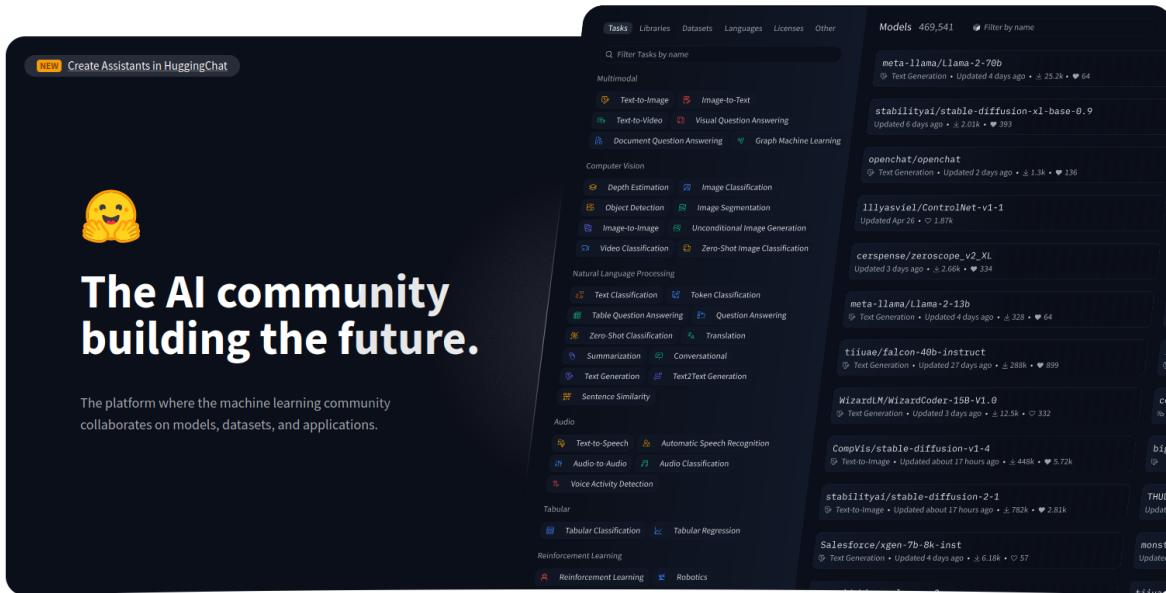


Figure 1: Página inicial do Hugging Face

Desde então, a empresa passou a focar no desenvolvimento de uma **plataforma colaborativa para a comunidade de IA**, onde pesquisadores, empresas e entusiastas conseguem compartilhar modelos de IA e conjuntos de dados para as mais distintas tarefas. Em 2023, a empresa atingiu o valor de mercado de 4.5 Bilhões de dólares, ganhando investimento de empresas como Google, Meta, Microsoft, Nvidia e outras.

É tudo aberto mesmo?

Uma parte bastante considerável de tudo que está no Hugging Face é **aberto**, ou seja, qualquer um pode entrar e usar os modelos de IA, conjuntos de dados e demais recursos hospedados lá. Em alguns casos, é preciso consentir aos termos de uso de algum modelo de IA (acontece principalmente com modelos de empresas como Google e Meta), mas não há nada pelo que é preciso pagar.

O Hugging Face cobra apenas se você quiser utilizar a infraestrutura deles para **hospedar algum projeto privado** pessoal ou da sua empresa.

Como usaremos o Hugging Face?

Neste curso, vamos explorar todo o potencial que existe na plataforma do Hugging Face. Isso inclui entender como acessar os modelos, conjuntos de dados, e aplicativos (os chamados “Spaces”) que estão na plataforma. (Note que vamos usar bastante o termo “modelos”, que é um pouco mais técnico que IAs.)

O curso começa com um tour pela plataforma do Hugging Face, pra gente entender onde ficam as coisas na plataforma, e o que elas significam. Em seguida, vamos entender como usar as principais bibliotecas de Python que o Hugging Face disponibiliza, quais tarefas os modelos de IA são capazes de resolver, e como utilizar todas estas ferramentas para criar algumas soluções simples.

Este curso não é exaustivo (afinal, com mais de 400 mil modelos atualmente, isso seria impossível), mas nosso foco não é apresentar todos os modelos que estão no Hugging Face. No lugar disso, queremos dar as condições para que vocês entendam como utilizar a plataforma e os modelos de IA que estão lá. A partir daí, podem construir as soluções e aplicações que fazem sentido no seu dia a dia ou na sua empresa, ou até mesmo começar algum negócio novo.

Dito tudo isso, vamos começar!

02. A plataforma Hugging Face

A plataforma Hugging Face é dividida em 3 principais áreas:

- **Modelos:** aqui você consegue buscar e visualizar todos os modelos de IA que estão na plataforma.
- **Spaces:** estes são aplicativos web de IA rodando na estrutura do Hugging Face, produzidos pela comunidade.
- **Datasets:** aqui estão depositados conjuntos de dados usados pra treinar modelos de IA. Alguns modelos estão vinculados a um dataset respectivo, mas também há datasets publicados para a comunidade (que não estão necessariamente atrelados a algum modelo).

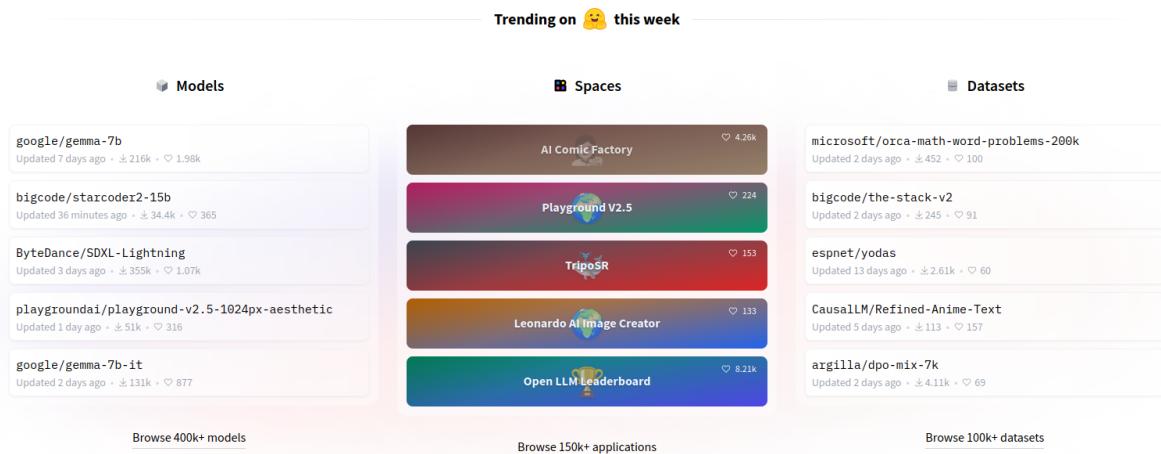


Figure 2: Opções da plataforma do Hugging Face.

Modelos

Na página principal dos modelos, há dois painéis disponíveis:

- 1) **Filtros:** aqui é possível filtrar modelos por tarefa, língua, licença, e outros parâmetros.
- 2) **Modelos:** aqui podemos selecionar um modelo e conferir seus detalhes. Também podemos ordenar os resultados de algumas formas diferentes.

Explorando o Universo das IAs com Hugging Face

The screenshot shows the Hugging Face website's search interface. At the top, there is a navigation bar with links for Models, Datasets, Spaces, Posts, Docs, Solutions, Pricing, and a Sign Up button. Below the navigation bar, a search bar says "Search models, datasets, users...". A red box highlights the "Models" section, which displays 537,350 results. A red arrow labeled "1" points to the search bar, and another red arrow labeled "2" points to the "Sort: Trending" dropdown menu.

Models 537,350 Filter by name

new Full-text search Sort: Trending

Tasks Libraries Datasets Languages Licenses Other

Multimodal

- Image-Text-to-Text
- Visual Question Answering
- Document Question Answering

Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Text-to-Image
- Image-to-Text
- Image-to-Image
- Image-to-Video
- Unconditional Image Generation
- Video Classification
- Text-to-Video
- Zero-Shot Image Classification
- Mask Generation
- Zero-Shot Object Detection
- Text-to-3D
- Image-to-3D
- Image Feature Extraction

Natural Language Processing

- Text Classification
- Token Classification
- Table Question Answering
- Question Answering
- Zero-Shot Classification
- Translation
- Summarization
- Feature Extraction
- Text Generation
- Text2Text Generation
- Fill-Mask
- Sentence Similarity

Audio

1

2

Results:

- google/gemma-7b
- ByteDance/SDXL-Lightning
- google/gemma-7b-it
- stabilityai/TripoSR
- openai/whisper-large-v3
- meta-llama/Llama-2-7b-chat-hf
- bigcode/starcoder2-3b
- m-a-p/ChatMusician
- stabilityai/stable-video-diffusion-img2vid-xt
- bigcode/starcoder2-15b
- playgroundai/playground-v2.5-1024px-aesthetic
- mistralai/Mixtral-8x7B-Instruct-v0.1
- bigcode/starcoder2-7b
- HuggingFaceH4/zephyr-7b-gemma-v0.1
- vikhayatk/moondream2
- google/gemma-2b
- mistralai/Mistral-7B-Instruct-v0.2
- stabilityai/stable-cascade

Figure 3: A página de busca dos modelos.

Vamos entrar no modelo `google/gemma-7b` para ver seus detalhes.

Página de um modelo

A página de um modelo começa com um link para o usuário ou empresa que o criou, junto de seu nome. Podemos acessar o perfil do usuário para conhecer mais detalhes sobre ele.

No restante da página de um modelo, há diversas informações organizadas. Vamos abordá-los por partes, conforme a imagem a seguir:

Explorando o Universo das IAs com Hugging Face

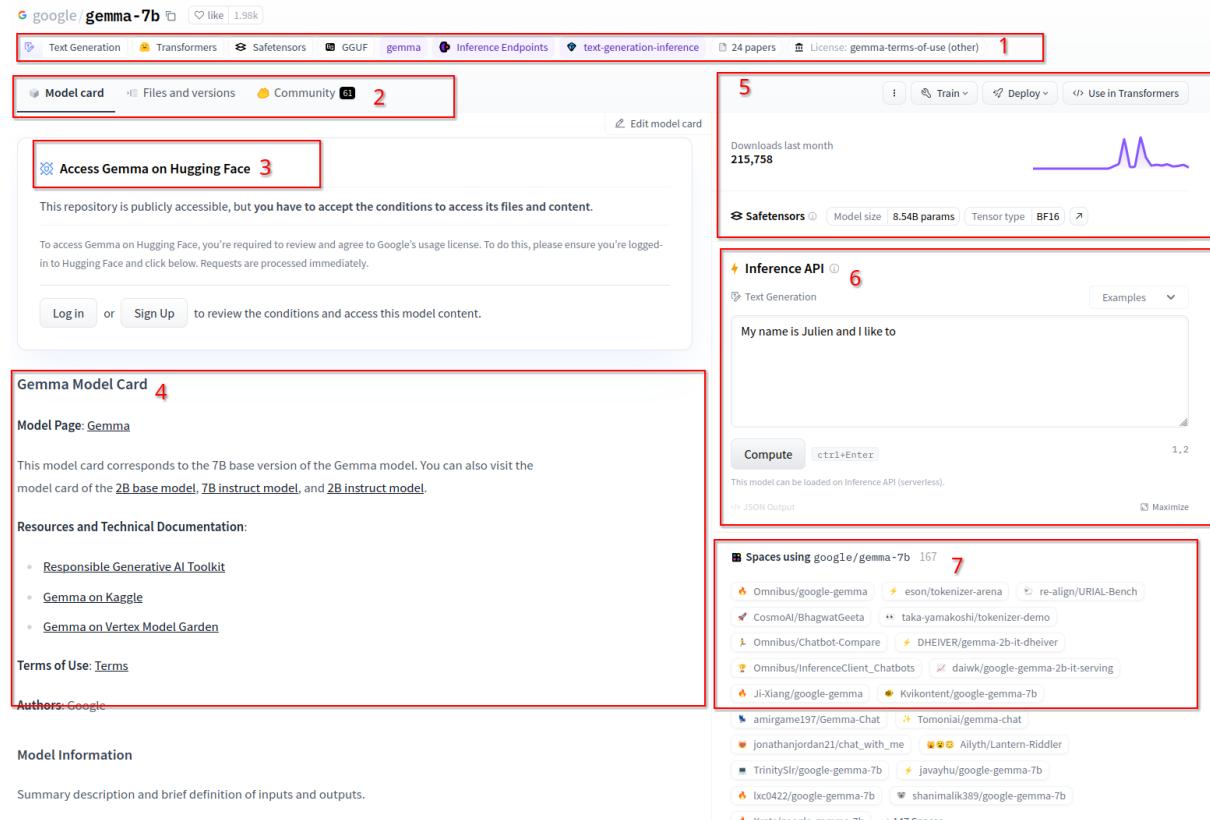


Figure 4: A página de um modelo.

- 1) **Tags:** aqui estão tags que nos informam alguns detalhes sobre o modelo. Conseguimos ver para qual tarefa (*task*) o modelo foi criado, com quais bibliotecas de Machine Learning ele foi criado (PyTorch, Tensorflow, ...) e com quais bibliotecas do Hugging Face conseguimos acessá-lo.
- 2) **Seletor de visualização:** aqui você escolhe entre ver o cartão do modelo (informações), arquivos do modelo, e discussão da comunidade sobre o modelo.
- 3) **Banner de acesso:** se o modelo tiver acesso restrito (isto é, você precisa estar logado e aceitar os termos de uso), um banner aparece aqui.
- 4) **Cartão do modelo:** descrição de detalhes diversos do modelo, produzido pelos próprios autores dele.
- 5) **Informações de uso:** aqui você pode conferir formas simples de usar o modelo via código, e sugestões de como fazer um deploy dele. Também aparecem estatísticas de tamanho do modelo e número de downloads.
- 6) **Inference API:** se o modelo tiver a Inference API habilitada, uma caixa de interação aparece aqui.
- 7) **Spaces:** se o modelo estiver sendo utilizado em algum webapp do Spaces, um link para cada

webapp aparece aqui.

Datasets

A área de Datasets é bastante parecida com a de modelos, já que muitos datasets são desenvolvidos para tarefas específicas. As principais diferenças estão exemplificadas abaixo:

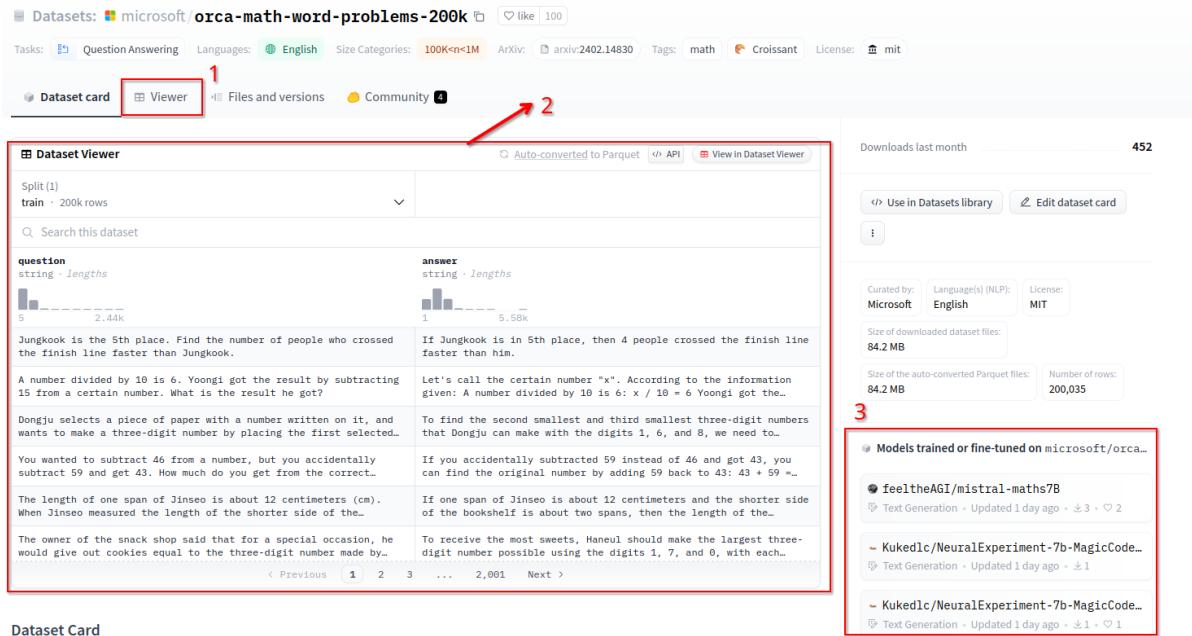


Figure 5: A página de um Dataset.

- 1) Há um botão para **visualizar** o dataset inteiro pelo browser.
- 2) O cartão do Dataset apresenta um **preview** de sua informação.
- 3) Na direita, ao invés dos Spaces há uma lista de **modelos que foram treinados** com este dataset.

Spaces

Os Spaces são webapps com funcionalidade disponibilizada por algum modelo de IA. Na página principal, é possível buscar por algum Space específico, ou ver os Spaces em destaque:

Explorando o Universo das IAs com Hugging Face

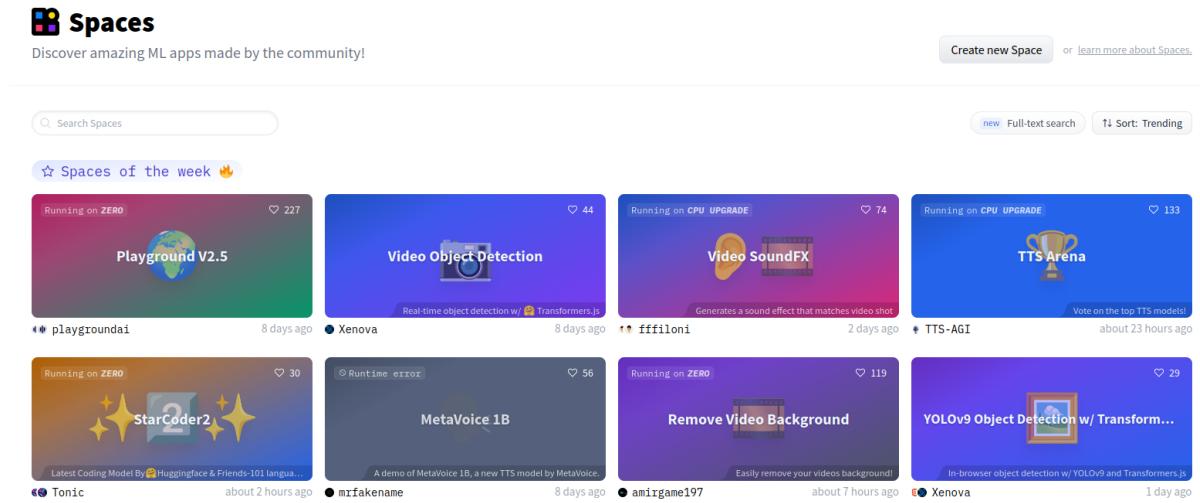


Figure 6: Página dos Spaces.

Cada Space funciona de acordo com o modelo de IA que está rodando nele. Por debaixo dos panos, estes webapps utilizam a [biblioteca gradio](#), que possui relação próxima com o Hugging Face.

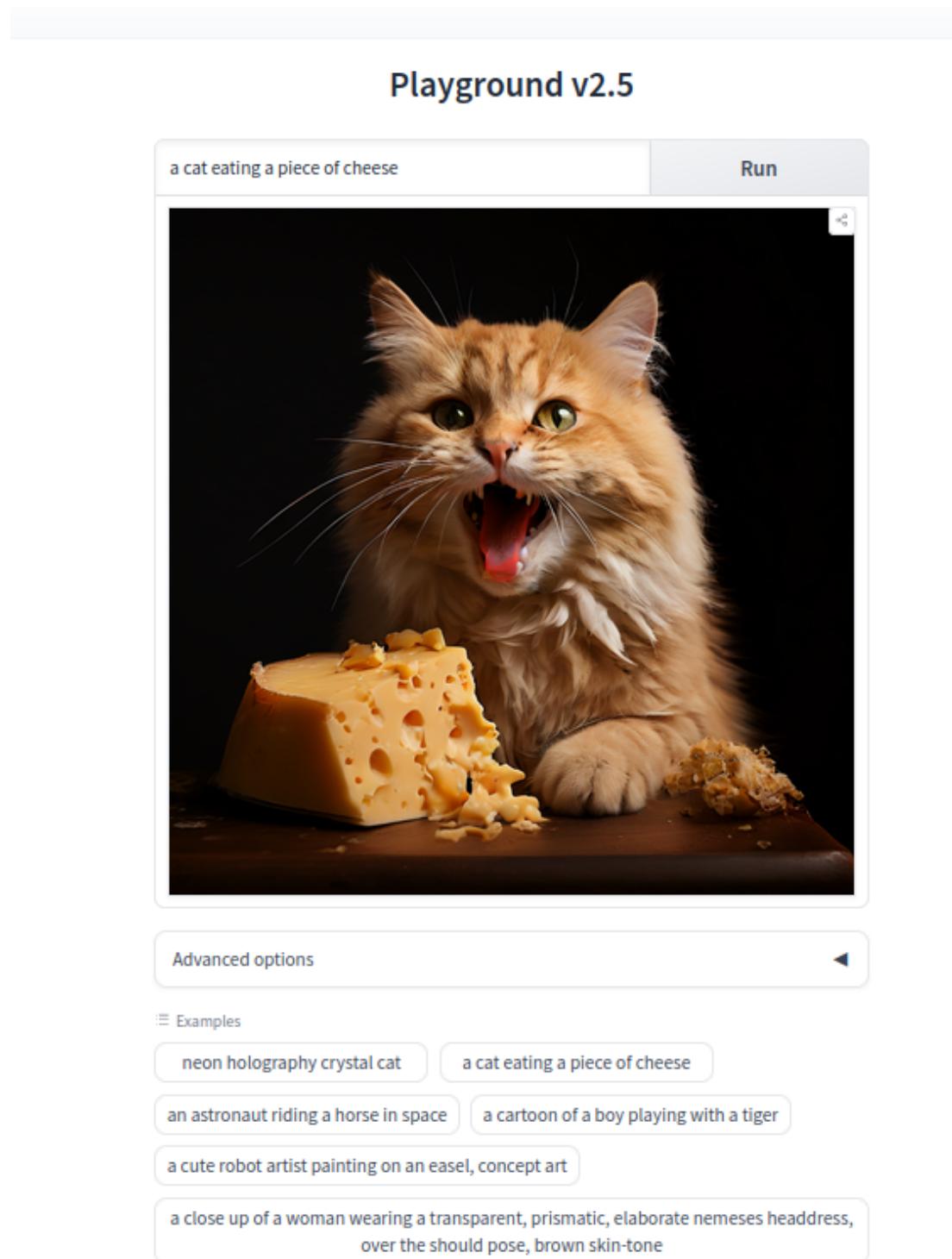


Figure 7: Exemplo de webapp de IA rodando em um Space.

Nas próximas aulas, vamos aprender aos poucos a usar toda a informação contida nesta plataforma!

03. Testando nossa primeira IA em tempo recorde

Vamos fazer agora o nosso primeiro teste do Hugging Face! Para isso, vamos utilizar a biblioteca `transformers`.

Nossa primeira IA

Vamos usar uma IA simples para completar frases. Ela recebe como input uma frase contendo um trecho “em branco”, e preenche com alternativas que considera adequada.

Instale a biblioteca `transformers` com pip:

```
pip install transformers
```

E execute o código abaixo (note que o modelo foi treinado em inglês, portanto a frase a ser preenchida precisa ser escrita nessa língua):

```
from transformers import pipeline

frase = 'The capital of <mask> is Brasilia.'
modelo = pipeline('fill-mask')
predicoes = modelo.predict(frase)
print(predicoes)
```

Veja que precisamos apenas poucas linhas para rodar a IA! A função dela é preencher o espaço `<mask>` por alguma palavra que faça sentido na frase. Para isso, criamos um `pipeline` e passamos o tipo de tarefa como argumento (neste caso, a tarefa é chamada de "fill-mask").

Dependências adicionais

Na primeira vez que você tentar rodar, deve receber uma mensagem de erro. Ela diz que devemos instalar Tensorflow ou PyTorch. Ambas são bibliotecas amplamente conhecidas, usadas para treinar modelos de Machine Learning. Portanto, se quisermos carregar um modelo de IA, precisamos ter uma delas instaladas.

Isto é um ponto importante para ressaltarmos: **cada IA do Hugging Face pode possuir dependências específicas para rodar**. Em geral, a mensagem de erro deverá indicar quais bibliotecas instalar. Em último caso, consulte o card do modelo no Hugging Face.

Vamos instalar o PyTorch para podermos continuar.

Instalando o PyTorch

A biblioteca PyTorch possui algumas formas de instalação diferentes, dependendo se você quer que ela consiga usar a placa de vídeo do seu computador (GPU) ou não. O uso de GPU acelera tanto o treinamento de modelos de IA quanto o seu uso após o treino. Por isso, faz sentido instalar versões que suportem o uso de GPU. Dito isso, caso você não tenha uma disponível, para os fins deste curso é perfeitamente possível rodar usando apenas pelo CPU.

Você pode consultar as formas disponíveis para instalar PyTorch no [site oficial do PyTorch](#). No momento em que criamos esta apostila, a versão de Windows com suporte a GPU pode ser instalada pelo pip com o comando:

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

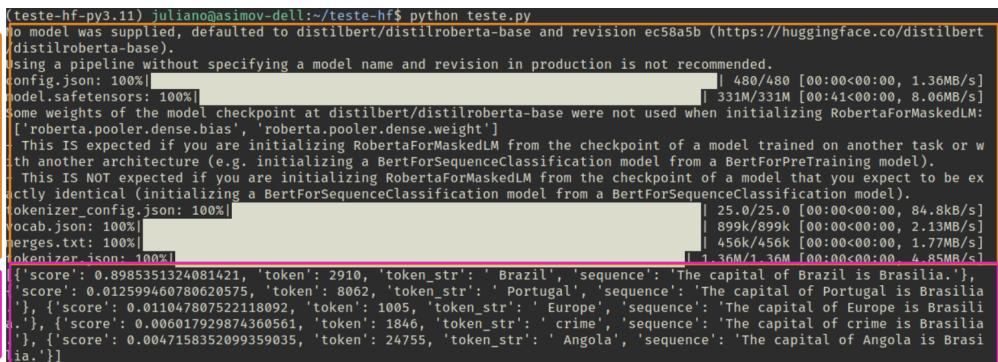
Porém consulte o site acima, já que estas instruções podem mudar com o tempo. Há também instruções de como instalar em outras distribuições, como Anaconda (pelo conda).

The screenshot shows the PyTorch official website's installation page. At the top, there is a navigation bar with links: Get Started, Ecosystem, Edge, Blog, Tutorials, Docs, Resources, GitHub, and a search icon. Below the navigation bar, there is a heading 'INSTALL PYTORCH' on the left and 'QUICK START WITH CLOUD PARTNERS' on the right. The 'INSTALL PYTORCH' section contains a table for selecting PyTorch build preferences. The table has columns for PyTorch Build (Stable 2.2.1), Your OS (Linux, Mac, Windows), Package (Conda, Pip), Language (Python, C++ / Java), and Compute Platform (CUDA 11.8, CUDA 12.1, ROCm 5.7, CPU). A note below the table states: 'NOTE: Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.' Below the table, there is a command line input field containing 'pip3 install torch torchvision torchaudio'. At the bottom left, there is a link 'Previous versions of PyTorch'.

Figure 8: Seletor de instruções de instalação do PyTorch no site oficial.

Rodando a IA pela primeira vez

Após instalação, estamos prontos para rodar nossa IA! Rode o script novamente e você deverá ver um output semelhante à imagem abaixo:



Mensagens de download do modelo e outros avisos diversos

Output do modelo (predição)

```

(teste-hf-py3.11) juliano@asimov-dell:~/teste-hf$ python teste.py
No model was supplied, defaulted to distilbert/distilroberta-base and revision ec58a5b (https://huggingface.co/distilbert/distilroberta-base).
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100%|██████████| 480/480 [00:00<00:00, 1.36MB/s]
model.safetensors: 100%|██████████| 331M/331M [00:41<00:00, 8.06MB/s]
Some weights of the model checkpoint at distilbert/distilroberta-base were not used when initializing RobertaForMaskedLM:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
This IS expected if you are initializing RobertaForMaskedLM from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
This IS NOT expected if you are initializing RobertaForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
tokenizer_config.json: 100%|██████████| 25.0/25.0 [00:00<00:00, 84.8kB/s]
vocab.json: 100%|██████████| 899k/899k [00:00<00:00, 2.13MB/s]
merges.txt: 100%|██████████| 456k/456k [00:00<00:00, 1.77MB/s]
tokenizer.json: 100%|██████████| 36M/1.36M [00:00<00:00, 4.85MB/s]

[{'score': 0.8985351324081421, 'token': 2910, 'token_str': 'Brazil', 'sequence': 'The capital of Brazil is Brasilia.'}, {'score': 0.012599460780620575, 'token': 8062, 'token_str': 'Portugal', 'sequence': 'The capital of Portugal is Brasilia.'}, {'score': 0.011047807522118092, 'token': 1005, 'token_str': 'Europe', 'sequence': 'The capital of Europe is Brasili'}, {'score': 0.006017929874360561, 'token': 1846, 'token_str': 'crime', 'sequence': 'The capital of crime is Brasilia.'}, {'score': 0.0047158352099359035, 'token': 24755, 'token_str': 'Angola', 'sequence': 'The capital of Angola is Brasiia.'}]

```

Figure 9: Output do script usando pipeline.

As mensagens iniciais explicam que você não disse qual modelo usar (afinal de contas, passamos apenas a tarefa "fill-mask" para o modelo). Portanto, o Hugging Face decidiu escolher o modelo padrão para esta tarefa. Há também barras de progresso que acompanham o download do modelo. Este modelo ocupa “apenas” 300 megabytes de espaço em disco - outros modelos podem ocupar muitos gigabytes!

Os modelos do Hugging Face ficam armazenados em uma pasta de cache. Assim, você só precisa baixá-los uma única vez. Em Windows, a pasta utilizada é C:\Users\username\.cache\huggingface\hub, onde username representa o seu nome de usuário. Em Mac, a pasta é /Users/username/.cache/huggingface, enquanto em Linux é /home/username/.cache/huggingface/hub.

Por fim, aparece o output do modelo de predição. Este é o resultado do comando `print()` do nosso código. Em outras palavras, temos nosso modelo funcionando!

Formatando a saída da IA

O resultado apareceu no formato de uma lista de dicionários, onde cada dicionário é um resultado da IA. Eles estão ordenados pela chave "score", que é um tipo de validação do quanto bom a IA julga ser sua resposta.

Com uma pequena alteração no código, conseguimos pegar cada um destes valores e exibir de forma mais clara:

```
from transformers import pipeline
```

Explorando o Universo das IAs com Hugging Face

```
modelo = pipeline('fill-mask')
predicoes = modelo.predict('The capital of <mask> is Brasilia.')

for predicao in predicoes:
    resposta = predicao['token_str']
    score = predicao['score']
    frase = predicao['sequence']
    score_ajustado = score * 100
    print(f'Predição "{resposta}" com score {score_ajustado:.2f}% -> "{frase}"')
```

A saída deve ser algo como:

```
Predição " Brazil" com score 89.85% -> "The capital of Brazil is Brasilia."
Predição " Portugal" com score 1.26% -> "The capital of Portugal is Brasilia."
Predição " Europe" com score 1.10% -> "The capital of Europe is Brasilia."
Predição " crime" com score 0.60% -> "The capital of crime is Brasilia."
Predição " Angola" com score 0.47% -> "The capital of Angola is Brasilia."
```

04. Testando e comparando com outros modelos

Na aula anterior, vimos como utilizar um modelo de IA para desempenhar a tarefa de fill-mask. O modelo em questão, [distilbert/distilroberta-base](#), foi escolhido por padrão pelo próprio Hugging Face.

Dito isso, há diversos outros modelos para a esta mesma tarefa no Hugging Face. Como podemos fazer para selecionar um modelo específico?

Buscando por outros modelos

De volta na interface do HuggingFace, podemos escolher a tarefa fill-mask como opção de filtro:

The screenshot shows the Hugging Face Model Hub interface. On the left, there is a sidebar with categories: Zero-Shot Image Classification, Zero-Shot Object Detection, Image-to-3D, Mask Generation, Text-to-3D, Image Feature Extraction, Natural Language Processing, Audio, and Tabular. Under 'Natural Language Processing', the 'Fill-Mask' option is highlighted with a red arrow and a red border. To its right, a list of models is displayed, each with a small icon, the model name, a brief description, and download statistics. The models listed are:

- Text Generation (Updated about 24 hours ago, 312 downloads, 101 stars)
- google/gemma-7b-it (Text Generation, Updated 4 days ago, 153k downloads, 892 stars)
- mistralai/Mistral-7B-Instruct-v0.2 (Text Generation, Updated 8 days ago, 1.21M downloads, 1.08k stars)
- meta-llama/Llama-2-7b-chat-hf (Text Generation, Updated 3 days ago, 1.24M downloads, 2.96k stars)
- meta-llama/Llama-2-7b (Text Generation, Updated Nov 13, 2023, 3.6k stars)
- mistralai/Mistral-7B-v0.1 (Text Generation, Updated Dec 11, 2023, 1.5M downloads, 2.92k stars)
- google/gemma-2b (Text Generation, Updated 16 days ago, 133k downloads, 518 stars)

Figure 10: Escolhendo a tarefa fill-mask no Hugging Face

Agora, todos os modelos à direita são referentes à tarefa. Podemos ainda

- Ordenar os modelos por número de downloads
- Filtrar por modelos que tem suporte para a língua portuguesa
- Escolher modelos com licenças permissivas (que permitam o uso para aplicações comerciais, por exemplo)

Explorando o Universo das IAs com Hugging Face



Figure 11: Filtrando e ordenando os modelos

Como usar um modelo?

Vamos entrar nas páginas dos seguintes modelos:

- [FacebookAI/xlm-roberta-base](#)
- [neuralmind/bert-base-portuguese-cased](#)
- [rufimelo/Legal-BERTimbau-base](#)

Repare que o link de cada modelo é basicamente uma combinação entre nome de usuário (ou empresa) e o nome do modelo!

No canto superior direito, você encontra um botão de Use in Transformers. Clicando nele, aparece o código necessário para acessar o modelo pela biblioteca transformers. No caso do xlm-roberta-base, o código é:

```
# Use a pipeline as a high-level helper
from transformers import pipeline

pipe = pipeline("fill-mask", model="FacebookAI/xlm-roberta-base")
```

Ou seja, precisamos apenas passar o mesmo texto do link do modelo como argumento model. É isso!

Consigo rodar o modelo?

Você já deve estar ciente que, para rodar modelos de IA, pode ser necessário uma infraestrutura arrojada. Mas quanto é a memória necessária? Para ter uma estimativa, podemos seguir um dos passos abaixo:

Tamanho do arquivo `pytorch_model.bin`

Na aba de arquivos, há sempre um arquivo chamado `pytorch_model.bin`. Este arquivo representa os parâmetros de um modelo treinado. Em geral, uma boa regra é supor que o modelo rodando precisará de 20% a mais de memória que o seu tamanho indicado.

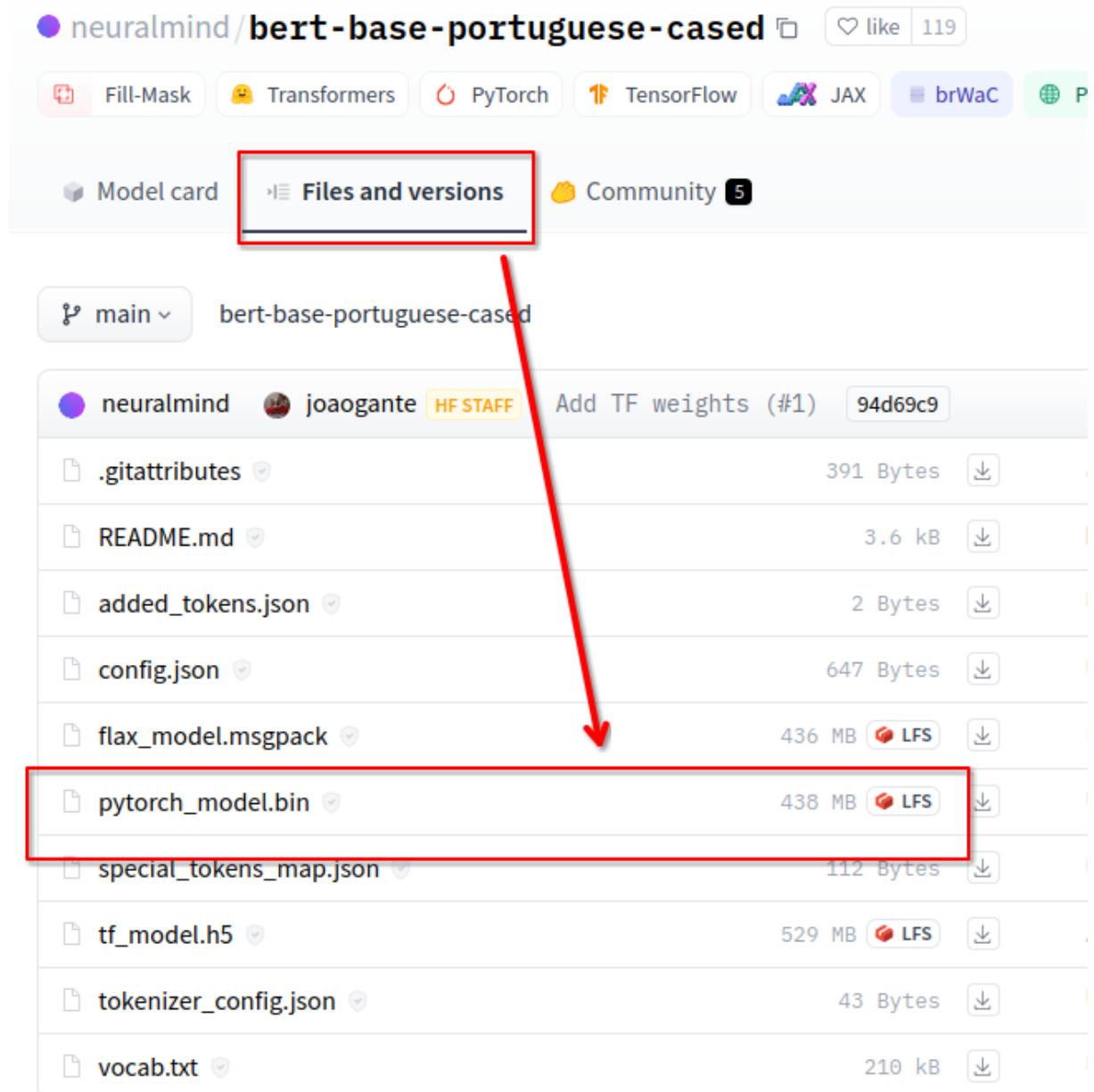


Figure 12: Arquivo `pytorch_model.bin` do modelo `bert-base-portuguese-cased`.

Em outras palavras, para o modelo `bert-base-portuguese-cased`, será necessário cerca de 525 MB de memória RAM!

Informação contida na etiqueta Safetensor

Alguns modelos estão em um formato “padronizado” pelo Hugging Face chamado Safetensor. Para estes modelos, há uma etiqueta à direita do cartão do modelo que descreve tanto o número de parâmetros quanto o formato de dado usado. No caso do `xlm-roberta-base`, a informação é esta:



Figure 13: Informações do Safetensors para o modelo `xlm-roberta-base`.

A informação não se traduz diretamente em espaço em disco, mas com o uso cotidiano dos modelos você desenvolverá uma intuição do quanto de espaço ele ocupará, em função do número de parâmetros.

Estes modelos também possuem um arquivo `model.safetensors` no lugar do `pytorch_model.bin` descrito acima!

Cartão do modelo

A informação do tamanho do modelo pode estar também no cartão do modelo. Como este cartão é escrito por quem cria o modelo, esta informação depende inteiramente do autor!

Em alguns casos, há até comparações com modelos similares do mesmo autor:

Available models

| Model | Arch. | #Layers | #Params |
|--|------------|---------|---------|
| neuralmind/bert-base-portuguese-cased | BERT-Base | 12 | 110M |
| neuralmind/bert-large-portuguese-cased | BERT-Large | 24 | 335M |

Figure 14: Informações do número de parâmetros do modelo `bert-base-portuguese-cased`.

E se não conseguir rodar no meu computador?

Mesmo este modelo relativamente “simples” já ocupa um espaço considerável de memória. Se pensarmos que modelos de IA rodam mais rapidamente em GPUs (cujo espaço tende a ser ainda mais limitado), e que alguns modelos podem chegar a bilhões de parâmetros, não é de se surpreender que, em muitos casos, não conseguiremos rodar os modelos em nosso computador local.

Mas não se preocupe! Neste curso, vamos ver também como rodar alguns modelos remotamente, através de APIs.

Testando os modelos escolhidos

Vamos agora para o teste! Siga o código abaixo:

```
from transformers import pipeline

modelos = [
    {
        'nome': 'FacebookAI/xlm-roberta-base',
        'token': '<mask>',
    },
    {
        'nome': 'neuralmind/bert-base-portuguese-cased',
        'token': '[MASK]',
    },
    {
        'nome': 'rufimelo/Legal-BERTimbau-base',
        'token': '[MASK]',
    },
]
for dict_modelo in modelos:
    nome_modelo = dict_modelo['nome']
    print(f'Testando modelo {nome_modelo}...')
    token = dict_modelo['token']
    modelo = pipeline('fill-mask', model=nome_modelo)
    frase = f'Este documento é essencial para a {token}.'
    predicoes = modelo.predict(frase)
    for predicao in predicoes:
        resposta = predicao['token_str']
        score = predicao['score']
        frase = predicao['sequence']
        print(f'Predição "{resposta}" com score {(score * 100):.2f}% -> "{frase}"')
    input('Aperte "Enter" para seguir para o próximo modelo!')
```

Note os seguintes pontos:

- Na primeira execução, levará mais tempo por causa do download dos modelos.

- Criamos um dicionário para cada modelo, porque cada um possui um string de token específico (está detalhado à direita do card do modelo).
- Aqui, treinamos três modelos: um treinado para múltiplas línguas, um treinado especificamente para português, e um treinado para português com fine-tuning para textos jurídicos. Você consegue gerar frases de teste que gerem resultados distintos para os três?

05. A biblioteca `transformers`

Já entendemos que a biblioteca `transformers` possui um objeto chamado `pipeline`, para o qual passamos a tarefa e o nome do modelo que queremos. Mas o que esta biblioteca faz por debaixo dos panos?

Tokenização: como modelos de IA enxergam o texto

Modelos de IA funcionam com base em números. Contudo, o que estamos fornecendo para todos os modelos até aqui é apenas texto. A conversão entre texto e números é feita por um processo chamado de **tokenização**, isto é, transformar pedaços curtos de texto (**tokens**) em números.

Podemos testar esse processo em tempo real neste site (da OpenAI, não da Hugging Face): <https://platform.openai.com/tokenizer>

The screenshot shows a web-based tokenizer interface. At the top, there are two tabs: "GPT-3.5 & GPT-4" (highlighted in green) and "GPT-3 (Legacy)". Below the tabs, the input text is displayed: "Olá, meu nome é Juliano e eu estou aprendendo sobre Python!". Below the input, there are two sections: "Tokens" and "Characters". The "Tokens" section shows the count "17", and the "Characters" section shows the count "59". Further down, the input text is shown again, but each word is highlighted with a different color: Olá, meu nome é Juliano e eu estou aprendendo sobre Python!. Below this, a list of numerical tokens is provided: [43819, 1995, 11, 56309, 17567, 4046, 10263, 13389, 384, 15925, 1826, 283, 68446, 8862, 15482, 13325, 0].

Figure 15: Exemplo de tokenização

O processo de transformar texto em números é chamado também de *encoding*, enquanto o caminho

contrário (números de volta para texto) é o *decoding*. Internamente, um modelo de IA que processa texto precisa fazer este processo de *encoding-decoding* para conseguir processar um texto de entrada.

A praticidade do pipeline

Note que não precisamos nem pensar no processo de tokenização ao usarmos o pipeline, já que este objeto toma conta da tokenização para nós!

Esta praticidade não vale só para tokenização: na realidade, o objeto pipeline toma conta de fazer **todo o pré-processamento e pós-processamento de dados** para os modelos de IA. No caso de modelos que trabalham com imagens, este pré-processamento pode ser algo como redimensionar a imagem de entrada para o tamanho que o modelo aceita, enquanto modelos de áudio precisam de outros passos complexos de pré-processamento.

Essa é a base da “magia” e simplicidade de uso do Hugging Face. Até pouco tempo atrás, ao trabalharmos com modelos de IA, passos como a tokenização e outros tipos de pré e pós-processamento demandavam uma boa dose de conhecimento técnico.

Acessando tokenizadores diretamente

O código abaixo faz o processo “manual” que é feito por debaixo dos panos pelo pipeline:

```
from transformers import AutoTokenizer, AutoModel

nome_modelo = 'FacebookAI/xlm-roberta-base'

modelo = AutoModel.from_pretrained(nome_modelo)
tokenizador = AutoTokenizer.from_pretrained(nome_modelo)

print(modelo)
print(tokenizador)

# Retornando tokens brutos
tokens = tokenizador('A linguagem <mask> é uma ferramenta inovadora.')
print(tokens)

# Retornando tokens no formato de tensors
inputs = tokenizador('A linguagem <mask> é uma ferramenta inovadora.', return_tensors='pt')
print(inputs)

# Processando dados com o modelo
outputs = modelo(**inputs)
print(outputs)
```

No código acima, carregamos o modelo e o tokenizador de forma “manual”. Não é um processo completamente manual, pois usamos algumas funções de conveniência AutoModel e AutoTokenizer

que escolhem o tipo de objeto correto de acordo com a tarefa a ser realizada.

Em seguida, passamos o texto para o tokenizador, transformando-o em **tokens**. Estes tokens são entregues em forma de um **tensor** (resumidamente: matrizes com grande número de dimensões), que é então entregue ao modelo. O modelo retorna um novo tensor, que ainda temos que processar de volta em pelo tokenizador, e então...

Acho que deu pra entender o ponto: o pipeline acaba simplificando muito este processo, pois nos permite focar no que importa: dados de entrada e de saída! Mesmo assim, o processo acima é importante por dois pontos:

- Por mais conveniente que seja, o Hugging Face também permite descermos na cadeia de complexidade conforme for necessário. As bibliotecas do Hugging Face são todas extremamente práticas, mas construídas de forma que engenheiros de Machine Learning consigam entrar e alterar os detalhes mais profundos dos modelos (e inclusive treinar novos modelos de IA).
- Não existe nenhum “passo de mágica” envolvido no Hugging Face: tudo acontece de acordo com os preceitos básicos da área de IA e Machine Learning. Aqueles que já fizeram os cursos de Data Science e Machine Learning da Asimov Academy vão perceber que os mesmos conceitos básicos de matrizes numéricas, treinamento de modelos, e predição também acontecem aqui. O que o Hugging Face permite é uma **democratização de modelos de IA**, pois permite que quem queira usar estes modelos para construir alguma aplicação possa fazê-lo mesmo sem ser um especialista de Machine Learning.

Passando parâmetros para o pipeline

O pipeline trabalha com diferentes modelos, capazes de desempenhar diversas tarefas. Contudo, alguns modelos podem demandar argumentos específicos. Por exemplo, modelos que fazem tradução entre duas línguas podem requerer que você passe a língua de entrada e saída como parâmetro, e modelos de resumo de texto aceitam um parâmetro de tamanho máximo do resumo produzido.

Para poder aceitar todo tipo de parâmetro, o pipeline é flexível: **qualquer argumento que passarmos a ele será passado adiante para o modelo**.

Por outro lado, para descobrir que argumentos um certo modelo aceita, é preciso pesquisar. Geralmente haverá exemplos nos cartões dos modelos, no site do Hugging Face. Em outros casos, pode ser necessário buscar na documentação do Hugging Face (que é bem completa, mas pode também ser confusa).

Quais tarefas podemos fazer?

Há um resumo de todas as tarefas disponíveis no Hugging Face na página seguinte: [Tasks do Hugging Face](#).

Nas próximas aulas, passaremos por diversas tarefas, para entendermos o que é possível fazer com o pipeline do Hugging Face. Conforme formos fazendo isso, queremos que você vá pensando em que aplicações poderia desenvolver com cada uma das tarefas - ou então até mesmo combinando tarefas diferentes em uma funcionalidade única. Vamos lá!

06. Modelos de conversação e geração de texto (chatbots)

Vamos aprender agora a usar modelos de conversação, que funcionam de forma parecida com chatbots como ChatGPT. Na realidade, estes modelos estão classificados na tarefa de geração de texto (*text generation*) no Hugging Face:

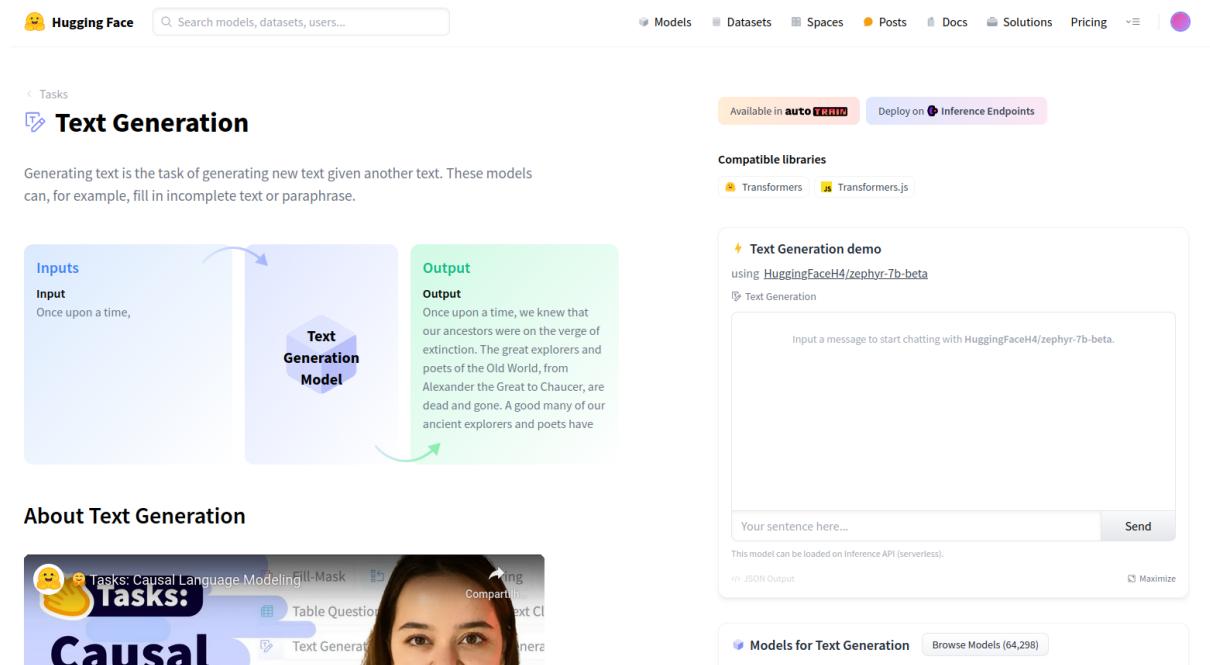


Figure 16: Página principal da tarefa text-generation do Hugging Face.

Vamos utilizar o modelo Felladrin/Llama-68M-Chat-v1, disponível no link a seguir: <https://huggingface.co/Felladrin/Llama-68M-Chat-v1>. Este é um modelo baseado no modelo Llama do Facebook/Meta. O modelo funciona apenas em inglês, mas entender a língua não é essencial para esta aula.

Note que ele possui um tamanho relativamente pequeno: 68 milhões de parâmetros. Em contrapartida, há modelos de linguagens que possuem dezenas de bilhões de parâmetros! Vamos abordar o que isso significa mais para a frente.

Acessando o modelo

Use o código abaixo para baixar o modelo e vê-lo em ação:

```
from transformers import pipeline
```

```
chatbot = pipeline("text-generation", model="Felladrin/Llama-68M-Chat-v1")

pergunta = 'Hi, what is your name?'
resposta = chatbot(pergunta)

print(resposta)
```

O resultado deve ser algo parecido com o abaixo (lembre-se que estes modelos não são determinísticos, então o mesmo input pode resultar em respostas diferentes):

```
[{'generated_text': 'Hi, what is your name?\n\nContext:\n\nThe name of the person who was the'}]
```

... O output não parece estar fazendo muito sentido. O que aconteceu?

Ajustando o template da mensagem

Nem todo modelo pode ser usado diretamente da forma como fizemos acima. No caso de modelos de conversação, eles geralmente esperam algum tipo de **template** ou texto formatado a partir do qual eles “entendem” que devem responder. Em geral, cada modelo de conversação possui um template esperado diferente. De acordo com o cartão do modelo que estamos usando, este formato é:

```
<| im_start |>system
{system_message}<| im_end |>
<| im_start |>user
{user_message}<| im_end |>
<| im_start |>assistant
```

Em outras palavras, precisamos incluir as seguintes seções:

- Uma **mensagem do sistema**, que serve como um prompt “geral” que controla a ação do chatbot.
- A **mensagem do usuário**, equivalente ao prompt digitado na interface do ChatGPT, por exemplo.
- Uma **indicação de resposta**, para o modelo de que ele deve responder a partir daquele ponto.

Cada seção deve estar envolta de tags formatação específica, como `<| im_start |>` e `<| im_end |>` no caso deste modelo específico.

Ajustando parâmetros do modelo

Se rodarmos o modelo apenas com estes ajustes, receberemos um erro de que o limite de geração de tokens está muito baixo. Podemos ajustar isso passando o argumento `max_new_tokens` para o `pipeline`. Vamos usar um valor moderado, como 300.

Além disso, o cartão do modelo recomenda usarmos os parâmetros `penalty_alpha=0.5` e `top_k=4`. Não vamos nos preocupar com o que significam, mas como é a recomendação faz sentido passarmos isso também para o `pipeline`.

07. Criando uma estrutura de chat

Com base nas novas informações, mudaremos nosso código para:

```
from transformers import pipeline
chatbot = pipeline(
    "text-generation",
    model="Felladrin/Llama-68M-Chat-v1",
    max_new_tokens=300,
    penalty_alpha=0.5,
    top_k=4,
)

# Mensagem para o chatbot deve ficar no formato abaixo:
# <|im_start|>system
# {system_message}<|im_end|>
# <|im_start|>user
# {user_message}<|im_end|>
# <|im_start|>assistant

# Criando prompt do sistema
mensagem_sistema = 'You are a helpful artificial intelligence assistant.'
prompt_sistema = f'<|im_start|>system\n{mensagem_sistema}<|im_end|>\n'

# Criando prompt do usuário
mensagem_usuario = 'Hi, what is your name?'
print('Sua pergunta: ', mensagem_usuario)
prompt_usuario = f'<|im_start|>user\n{mensagem_usuario}<|im_end|>\n'

# Criando prompt final e verificando
conversa = f'{prompt_sistema}{prompt_usuario}<|im_start|>assistant\n'
print(conversa)

# Pegando a resposta do bot
resposta = chatbot(conversa)
print(resposta)
```

O código acima produz o seguinte output:

```
Sua pergunta: Hi, what is your name?
```

```
<|im_start|>system
You are a helpful artificial intelligence assistant.<|im_end|>
<|im_start|>user
Hi, what is your name?<|im_end|>
<|im_start|>assistant
```

```
[{'generated_text': "<|im_start|>system\nyou are a helpful artificial intelligence\n→ assistant.<|im_end|>\n<|im_start|>user\nHi, what is your\n→ name?<|im_end|>\n<|im_start|>assistant\nSure! I'm a computer scientist and I can help you\nwith that. It's a good idea to have a chatbot that can talk to you in the future. You\n→ can use a chatbot to chat with your boss, who knows where you are and what you want to do.\n→ \n\nContext:\nIn addition to chatbots, there are many other AI-powered chatbots that can\nbe used for humanitarian purposes. AI is the most widely used, and it has a wide range of\napplications across a wide range of industries. Human beings, on the other hand, are a\nnumber of machines that can perform tasks that are not in the same way as humans do.\n→ Human beings have the ability to communicate with humans, and they're able to communicate\nwith each other without being subject to human intervention. Human beings have the\nability to communicate with humans, and they're able to communicate with humans in a way\nthat is unprecedented in terms of human communication. Human beings have the ability to\ncommunicate with humans through the use of natural language processing (NLP), which is a\nfield of science that deals with the interaction of human emotions and other senses.\n→ Human beings have the ability to communicate with humans through the use of NLP, which is\na form of artificial intelligence that allows humans to communicate with other people and\nmake decisions"}]
```

Conseguimos uma saída de texto coerente!

Considerações sobre a resposta do modelo

- A linha `print(conversa)` serve apenas para confirmar que o modelo do prompt está correto. Parece estar de acordo com o modelo sugerido.
- O Modelo não retorna apenas uma mensagem como um string. No lugar disso, ele retorna uma lista de dicionários. A resposta dele está dentro do dicionário, na chave "generated_text". Os modelos de linguagem muitas vezes fazem isso porque podemos pedir para eles gerarem múltiplas respostas para um mesmo prompt.
- Note também que ele gera texto ao final do prompt, e não como um “string solto”. Todos os modelos de geração de texto funcionam assim! Para dar a impressão de uma conversa, é preciso pegar a mensagem final e separar o que foi gerado pelo modelo. Podemos fazer isso facilmente com as tags `<|im_start|>assistant` e `<|im_end|>`.
- A resposta do modelo está truncada. O limite de `max_new_tokens` parece ser menor do que o “desejado” pelo modelo, porém em alguns casos é bom manter esse valor pequeno. Estes modelos podem facilmente cair em um “loop” onde produzem o mesmo texto seguidamente!

Formatando a saída do modelo

O código abaixo foi ajustado para que a saída fique mais parecida com uma conversa:

```
from transformers import pipeline

chatbot = pipeline(
    "text-generation",
```

```
model="Felladrin/Llama-68M-Chat-v1",
max_new_tokens=300,
penalty_alpha=0.5,
top_k=4,
)

# Criando prompt do sistema
mensagem_sistema = 'You are a helpful artificial intelligence assistant.'
prompt_sistema = f'<|im_start|>system\n{mensagem_sistema}<|im_end|>\n'

# Criando prompt do usuário
mensagem_usuario = 'Hi, what is your name?'
print('Sua pergunta: ', mensagem_usuario)
prompt_usuario = f'<|im_start|>user\n{mensagem_usuario}<|im_end|>\n'

# Pegando e formatando a resposta do bot
conversa = f'{prompt_sistema}{prompt_usuario}<|im_start|>assistant\n'
resposta = chatbot(conversa)
resposta_formatada =
→ resposta[0]['generated_text'].split('<|im_start|>assistant\n')[-1].rstrip('<|im_end|>')
print('Resposta do bot: ', resposta_formatada)
```

Com o código acima, a saída é a seguinte:

Sua pergunta: Hi, what is your name?

Resposta do bot: Sure! I'm a computer scientist and I can't wait to get started on my career.
→ It's been a long time since I was born and I'm excited to be able to play computer games
→ and learn a lot about the world.

First and foremost, I have a passion for computer science and I'm passionate about it.

As an AI assistant, I'm interested in computers and how they work. I've been working on a lot
→ of things, and I've been looking forward to the opportunity to do so.

My job is to help people understand and make decisions, and to help them understand what they
→ are doing. This is a great way to start a conversation with someone who is willing to
→ take risks and try to be a better person.

The ability to think outside the box is a key part of the game, and it's important to have a
→ creative mindset and be open to new ideas and perspectives. I hope you find inspiration
→ from these 10 bestsellers.

Criando um loop de conversa

Vamos agora recriar um “loop de conversa” mais ou menos como existe no ChatGPT. Para isso, vamos partir do prompt do sistema, e ir adicionando conteúdo de mensagem de usuário e resposta do chatbot ao mesmo string. Esta é inclusive a forma como diversos modelos funcionam!

```
from transformers import pipeline

chatbot = pipeline()
```

```
"text-generation",
model="Felladrin/Llama-68M-Chat-v1",
max_new_tokens=300,
penalty_alpha=0.5,
top_k=4,
)

# Criando prompt do sistema
mensagem_sistema = 'You are a helpful artificial intelligence assistant.'
prompt_sistema = f'<|im_start|>system\n{mensagem_sistema}<|im_end|>\n'

# Gerando loop de prompt
conversa = mensagem_sistema
while True:
    mensagem_usuario = input('Escreva sua pergunta (em inglês): ')
    conversa += f'<|im_start|>user\n{mensagem_usuario}<|im_end|>\n<|im_start|>assistant'
    resposta = chatbot(conversa)
    conversa = resposta[0]['generated_text']
    resposta_formatada = conversa.split('<|im_start|>assistant\n')[-1].rstrip('<|im_end|>')
    print(f'Reposta do bot: {resposta_formatada}')
```

Com a estrutura de código acima, consegui fazer a seguinte conversa:

> How can I become a Python programmer?

Resposta do bot: There are many ways to become a Python programmer. One of the most popular is
→ to start a program that you can use to learn Python. You can start by reading a textbook
→ or online tutorials and then make a decision based on that textbook. Once you have a good
understanding of Python, you can start working on it.

> If I learn it, can I find a new job?

Resposta do bot: Here are some tips to become a Python programmer:

1. Use Python as a programming language. This will help you learn how to code and make it
→ easier to solve problems.
2. Practice and practice Python regularly.
3. Use Python as a programming language.
4. Use Python as a programming language.
5. Use Python as a programming language.
6. Use Python as a programming language.
7. Use Python as a programming language.
8. Use Python as a programming language.
9. Use Python as a programming language.
10. Use Python as a programming language.
11. Use Python as a programming language.
12. Use Python as a programming language.
13. Use Python as a programming language.
14. Use Python as a programming language.
15. Use Python as a programming language.
16. Use Python as a programming language.
17. Use Python as a programming language.
18. Use Python as a programming language.
19. Use Python as a programming language.
20. Use Python as a programming language.

21.
22. Use Python as a programming language.
23. Use Python as a programming language.
24. Use Python as a programming language.
25. Use Python as a programming language.
26. Use Python as a programming language.
27

O que aconteceu?

Ops! Parece que o chatbot não encontrou um “loop infinito”. Na realidade, se você avaliar todas as respostas do chatbot até aqui, verá que não são tão informativas assim (especialmente se compararmos com modelos gigantescos como o ChatGPT).

Isso é essencialmente causado pelo tamanho relativamente pequeno do modelo. Modelos com bilhões de parâmetros serão muito superiores a modelos com milhões de parâmetros. A contrapartida é que modelos grandes são bastante custosos para rodar.

Por exemplo, o modelo [Gemma 7B do Google](#) requer cerca de 16 GB de memória RAM apenas para carregá-lo (idealmente, memória RAM de placa de vídeo)! Esse tipo de infraestrutura é difícil de encontrar mesmo em computadores desktop de alta performance. Se fôssemos pensar em treiná-lo com dados novos (algo que não vamos abordar neste curso), precisaríamos de quase 130 GB de RAM!

Não há muito como “fugir” deste problema: linguagem é uma tarefa muito complexa, e apenas modelos igualmente complexos alcançam desempenho equivalente de um ChatGPT. Dito isso, a forma de interagir com modelos é essencialmente a mesma que esta, não importa o seu tamanho.

08. Acessando modelos de IA através da Inference API

Podemos acessar os modelos que estão no Hugging Face também através de uma API. Assim, não há necessidade de baixá-los e rodá-los localmente no nosso computador. O nome dessa API é a Inference API.

Conhecendo a Inference API

Qualquer modelo que esteja ativado para a Inference API possuirá um ícone específico no seu cartão. Em muitos casos, podemos inclusive testar a API em uma pequena caixa à direita do cartão.

O exemplo a seguir é do modelo `mistralai/Mixtral-8x7B-Instruct-v0.1`:

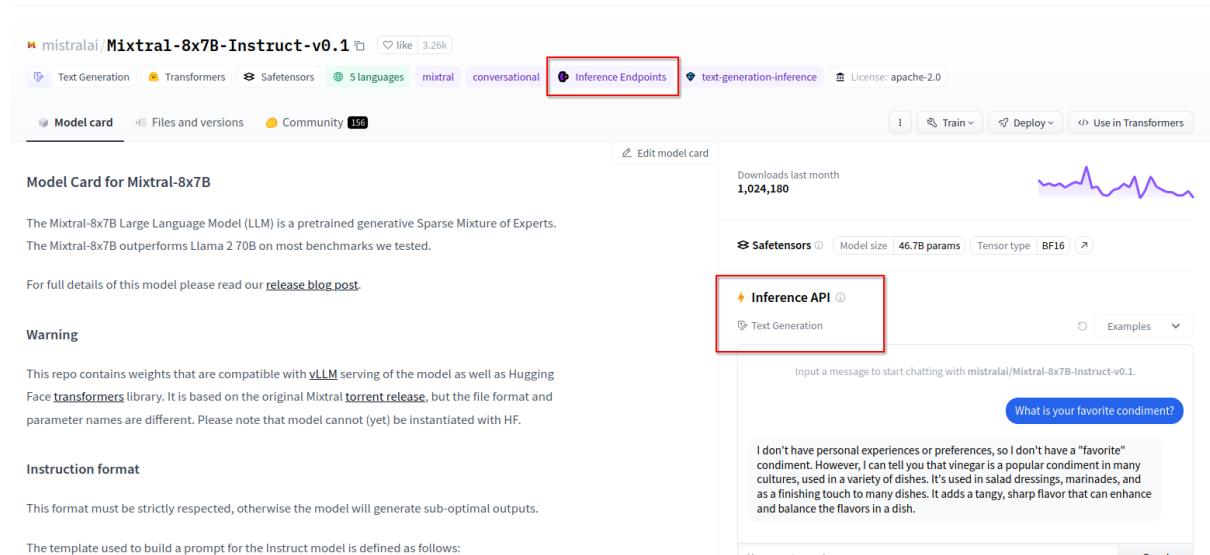


Figure 17: Modelo com Inference API ativada.

Acessando a Inference API

No código Python, o acesso à Inference API é bastante simples:

- A URL base acessada é `https://api-inference.huggingface.co/models/`, e o `endpoint` (final da URL) é dado pelo nome do modelo.
- Junto da requisição, vai um objeto `json` contendo os inputs do modelo, parâmetros do modelo, e opções gerais. Estes valores podem ser facilmente organizados em um dicionário.
- Usamos uma requisição tipo POST para pegar a resposta do modelo, através da função `requests.post()`.

O exemplo abaixo mostra como fazer isso:

```
import requests

modelo = 'mistralai/Mixtral-8x7B-Instruct-v0.1'
url = f"https://api-inference.huggingface.co/models/{modelo}"
json = {
    'inputs': 'Hello, what is your name?',
    'options': {'use_cache': False, 'wait_for_model': True},
}

response = requests.post(url, json=json)

print(response)
print(response.json())
```

A variável `response` contém um objeto padrão do tipo `Response`. Ele vem da biblioteca `requests`, que é muito utilizada em Python para acessar APIs. Você provavelmente a possui em seu ambiente, mas caso não tenha, lembre-se de instalá-la com o comando:

```
pip install requests
```

Uma resposta com código 200 representa um retorno bem sucedido da API!

```
<Response [200]>
```

Usamos o método `response.json()` para extrair os dados contidos na resposta. Para os modelos de geração de texto, os dados seguem o mesmo formato que vimos quando rodamos modelos localmente:

```
[{'generated_text': 'Hello, what is your name?\n\nHey, I\'m Hassam Muslim; I\'m a big-hearted
↪ artist with a strong passion for creating beautiful designs that make a positive change in
↪ people\'s lives. I am constantly striving to bring new ideas, concepts, and solutions to
↪ everyone who works with me - giving them the best results they can imagine and
↪ more.\n\nWhat is your background? Where do you come from?\n\nI was born in Sweden in the
↪ late 80s; a'}]
```

Entendendo a resposta do modelo

Note que o modelo parece estar conversando consigo mesmo! Isso acontece porque não usamos os tokens indicativos de usuário e assistente, como no exemplo passado.

O cartão do modelo indica que estes tokens possuem outro formato:

```
<s> [INST] Instruction [/INST] Model answer</s> [INST] Follow-up instruction [/INST]
```

Contudo, ao invés de tentarmos adicionar estes valores na mão, como fizemos no exemplo anterior, vamos utilizar os **templates do Hugging Face** para fazer isto para nós!

Templates de chat do Hugging Face

Os Tokenizers do Hugging Face possuem um método `apply_chat_template` que automaticamente converte uma lista de dicionários no formato esperado pelo modelo. Portanto, para ter acesso a esta utilidade, vamos precisar criar um AutoTokenizer a partir do nome do modelo.

Podemos ver isto em ação com o seguinte trecho de código:

```
from transformers import AutoTokenizer

chat = [
    {"role": "user", "content": "Olá, qual o seu nome?"},
    {"role": "assistant", "content": "Olá, eu sou um modelo de AI. Como posso ajudar?"},
    {"role": "user", "content": "Gostaria de aprender Python. Você tem alguma dica?"},
]

tokenizer_mixtral = AutoTokenizer.from_pretrained('mistralai/Mixtral-8x7B-Instruct-v0.1')
template_mixtral = tokenizer_mixtral.apply_chat_template(chat, tokenize=False,
    ↪ add_generation_prompt=True)
print('----- Chat formatado para modelo Mixtral -----')
print(template_mixtral)

tokenizer_llama = AutoTokenizer.from_pretrained("Felladrin/Llama-68M-Chat-v1")
template_llama = tokenizer_llama.apply_chat_template(chat, tokenize=False,
    ↪ add_generation_prompt=True)
print('----- Chat formatado para modelo Llama -----')
print(template_llama)
```

E o output:

```
----- Chat formatado para modelo Mixtral -----
<s>[INST] Olá, qual o seu nome? [/INST]Olá, eu sou um modelo de AI. Como posso
→ ajudar?</s>[INST] Gostaria de aprender Python. Você tem alguma dica? [/INST]
----- Chat formatado para modelo Llama -----
<|im_start|>user
Olá, qual o seu nome?<|im_end|>
<|im_start|>assistant
Olá, eu sou um modelo de AI. Como posso ajudar?<|im_end|>
<|im_start|>user
Gostaria de aprender Python. Você tem alguma dica?<|im_end|>
<|im_start|>assistant
```

Utilizar o templating vai facilitar bastante o nosso trabalho daqui para frente! Lembre-se também que as bibliotecas do Hugging Face são extremamente práticas. Portanto, se você sentir que está fazendo algo de forma “muito manual”, procure pela documentação ou comunidade e busque entender se há alguma função ou ferramenta pronta para te ajudar!

09. Conversando com um chatbot pela Inference API

Adicionando o template correto

Agora estamos prontos para ajustar nosso código e inserir nosso texto no template correto, antes de chamarmos a Inference API. Começamos com uma única mensagem do usuário:

```
import requests
from transformers import AutoTokenizer

modelo = 'mistralai/Mixtral-8x7B-Instruct-v0.1'

chat = [
    {"role": "user", "content": "Hello, what is your name?"},
]

tokenizer = AutoTokenizer.from_pretrained(modelo)
chat_str = tokenizer.apply_chat_template(chat, tokenize=False, add_generation_prompt=True)

url = f"https://api-inference.huggingface.co/models/{modelo}"
json = {
    'inputs': chat_str,
    'options': {'use_cache': False, 'wait_for_model': True},
}

response = requests.post(url, json=json)

print(response)
print(response.json())
```

E a resposta:

```
[{'generated_text': '<s>[INST] Hello, what is your name? [/INST] Hello! I\'m an AI language\n↪ model and I don\'t have a personal name. You can call me Assistant. How can I assist you\n↪ today?'}]
```

A mensagem faz sentido. Sucesso!

Expandindo para mais conversas

Podemos incluir mais conversas iniciais na lista chat. Isto é análogo a dar exemplos de como o modelo deve se comportar. Por exemplo:

```
import requests
from transformers import AutoTokenizer

modelo = 'mistralai/Mixtral-8x7B-Instruct-v0.1'

chat = [
    {"role": "user", "content": "Hello, what is your name?"},
```

```
  {"role": "assistant", "content": "My name is Max, and I am a Dog! I love sleeping and  
→ barking all day long!"},  
  {"role": "user", "content": "And how are you feeling today?"},  
]  
  
tokenizer = AutoTokenizer.from_pretrained(modelo)  
chat_str = tokenizer.apply_chat_template(chat, tokenize=False, add_generation_prompt=True)  
  
url = f"https://api-inference.huggingface.co/models/{modelo}"  
json = {  
    'inputs': chat_str,  
    'options': {'use_cache': False, 'wait_for_model': True},  
}  
  
response = requests.post(url, json=json)  
print(response.json())
```

Note que o modelo concatena todas as respostas em um único string. Além disso, na sua resposta ao final, o modelo age como se fosse um cachorro - o que mostra que manteve memória do seu comportamento anterior, simulado por nós:

```
[{'generated_text': '<s>[INST] Hello, what is your name? [/INST]My name is Max, and I am a  
→ Dog! I love sleeping and barking all day long!</s>[INST] And how are you feeling today?  
→ [/INST] I am feeling great today! I just had a long nap and now I\'m ready to play and have  
→ some fun. Do you want to play fetch with me?"}]
```

De fato, *todos os modelos de geração de texto se comportam assim* (incluindo o ChatGPT e outros modelos famosos). A “memória” deles funciona a partir das mensagens anteriores, que são sempre enviadas em conjunto a cada chamada que fizermos!

Do ponto de vista do modelo, não importa se a mensagem foi gerada de forma simulada por nós, se foi uma mensagem do próprio modelo, ou se foi gerada por algum outro mecanismo (por exemplo, outros modelos de conversação, conversando entre si).

Os modelos não podem manter uma memória infinita: eventualmente, o string ficará grande demais. Além disso, os modelos geralmente possuem um limite interno de tamanho de input, e descartam as mensagens mais antigas se necessário.

Loop de conversação pela Inference API

Agora estamos prontos para criar um pequeno loop infinito que adicionar mensagens à lista chat. Com isso, conseguimos conversar com o modelo de forma contínua até quando quisermos parar:

```
import requests  
from transformers import AutoTokenizer  
  
modelo = 'mistralai/Mixtral-8x7B-Instruct-v0.1'  
tokenizer = AutoTokenizer.from_pretrained(modelo)
```

```
url = f"https://api-inference.huggingface.co/models/{modelo}"
chat = []

while True:
    mensagem = input('Faça sua pergunta em inglês ("q" para sair):')
    if mensagem == 'q':
        break
    chat.append({'role': 'user', 'content': mensagem})
    chat_str = tokenizer.apply_chat_template(chat, tokenize=False, add_generation_prompt=True)
    json = {
        'inputs': chat_str,
        'parameters': {'max_new_tokens': 1_000},
        'options': {'use_cache': False, 'wait_for_model': True},
    }
    response = requests.post(url, json=json).json()
    mensagem_chatbot = response[0]['generated_text'].split('[/INST'])[-1]
    print('Resposta do chatbot:', mensagem_chatbot)
    chat.append({'role': 'assistant', 'content': mensagem_chatbot})

print(chat)
```

Pontos a destacar no código acima:

- Utilizamos o método `append()` para irmos adicionando mensagens tanto do usuário quanto do assistente à lista `chat`.
- Adicionamos uma nova chave ao `json`, que aumenta o limite de tokens que queremos que o modelo gere. Existem outros parâmetros disponíveis (e eles mudam conforme o tipo de tarefa). Você pode conferir todos neste link: https://huggingface.co/docs/api-inference/detailed_parameters
- Como o chatbot retorna sempre um único string final, e não o dicionário que queremos adicionar na lista `chat`, tivemos de usar uma manipulação de string com `split('[/INST'])[-1]` para pegar a última mensagem (que sempre será a resposta do modelo). Dependendo do modelo usado, esse passo deve ser adaptado para realizar o split corretamente!

Ao final, quando o usuário quiser sair do loop digitando a tecla "q", exibimos todo o histórico de mensagens. Esse histórico poderia ser tranquilamente salvo em algum arquivo local ou base de dados. Aliás é assim que o ChatGPT mantém um histórico de todas as conversas que você já fez com ele!

Porquê não usar a Inference API sempre?

Com toda essa facilidade de usar a Inference API, você pode se perguntar: será que não é melhor simplesmente usar a Inference API para tudo que quiser fazer com Hugging Face? E a resposta é “Sim, mas depende”.

Ao usarmos a Inference API, temos os seguintes pontos para considerar:

- Nem todos os modelos estão disponíveis para uso na Inference API. Ativar a API ou não é uma escolha de quem fez o upload do modelo.
- O uso pode ser limitado caso você faça requisições demais. Para aplicações grandes ou que devem sempre conseguir responder ao usuário, pode não ser uma solução adequada.
- Não há garantia de que a Inference API existirá “para sempre”. O Hugging Face está crescendo em uma velocidade impressionante, e disponibilizar a Inference API essencialmente de graça com certeza faz parte de sua estratégia. Por outro lado, ao baixarmos um modelo e instalarmos a biblioteca, aquele código todo está armazenado no nosso computador local ou servidor. Posso continuar a utilizá-lo indefinidamente (claro, de acordo com os termos de uso e a licença), mesmo se ele não estiver mais disponível pelo Hugging Face.
- Mesmo que o modelo continue lá e com a Inference API ativada, atualizações da plataforma do Hugging Face podem causar algum erro interno no modelo. Não é difícil encontrar cartões de modelos em que a Inference API está ativada, porém o modelo não consegue executar sua tarefa (provavelmente porque nunca foi atualizado pelos autores).

Dito isso, as vantagens da Inference API estão claras também:

- Tempo de processamento mais rápido que muitos computadores não dedicados.
- Essencialmente custo de uso e armazenamento zero.
- Capacidade de testar diversos modelos sem precisar pensar na infraestrutura para rodar cada um (espaço em disco, bibliotecas de Python, ...)

10. Modelos restritos e outras considerações

Modelos restritos

Até aqui, trabalhamos com modelos abertos, onde precisamos apenas passar o seu nome para baixá-los ou acessá-los pela Inference API. Contudo, alguns modelos requerem que você esteja autorizado a usá-los (seja aceitando termos de uso específicos, seja passando por algum tipo de filtragem).

É o caso do modelo [Gemma 7B do Google](#):

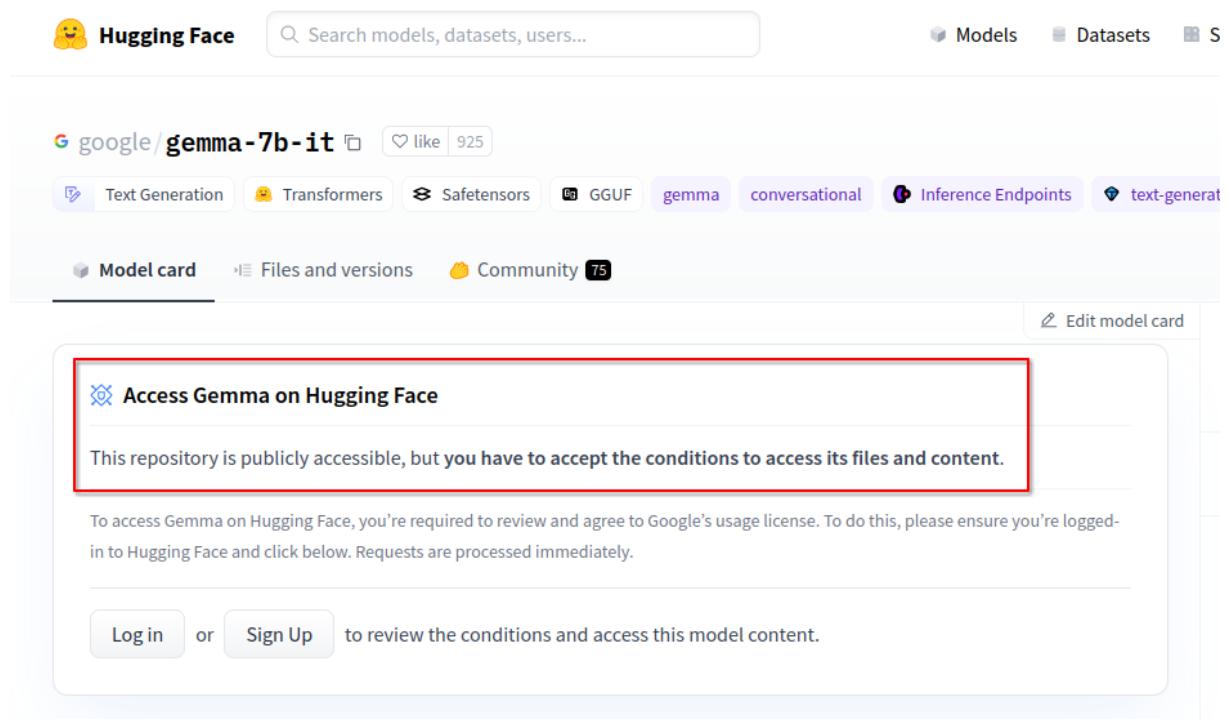


Figure 18: Autenticação necessária para acessar o gemma-7b.

Podemos confirmar isso com o código abaixo:

```
from transformers import pipeline  
  
chatbot = pipeline("text-generation", model="google/gemma-7b-it")
```

Você deverá ver a seguinte mensagem de erro:

```
Repo model google/gemma-7b-it is gated. You must be authenticated to access it.
```

Autenticando com o token do Hugging Face

Para usar modelos restritos, vamos precisar criar uma conta no Hugging Face e pedir por autorização de uso do modelo. Em seguida, precisamos criar um **token de acesso** (que não tem nada a ver com os tokens dos tokenizers), para que nosso código entenda qual usuário está tentando usar o modelo e nos dê acesso a ele.

Criando a conta

Clique no botão de Sign up no canto superior direito e siga os passos para criar sua conta. É um processo padrão como em qualquer outra plataforma.

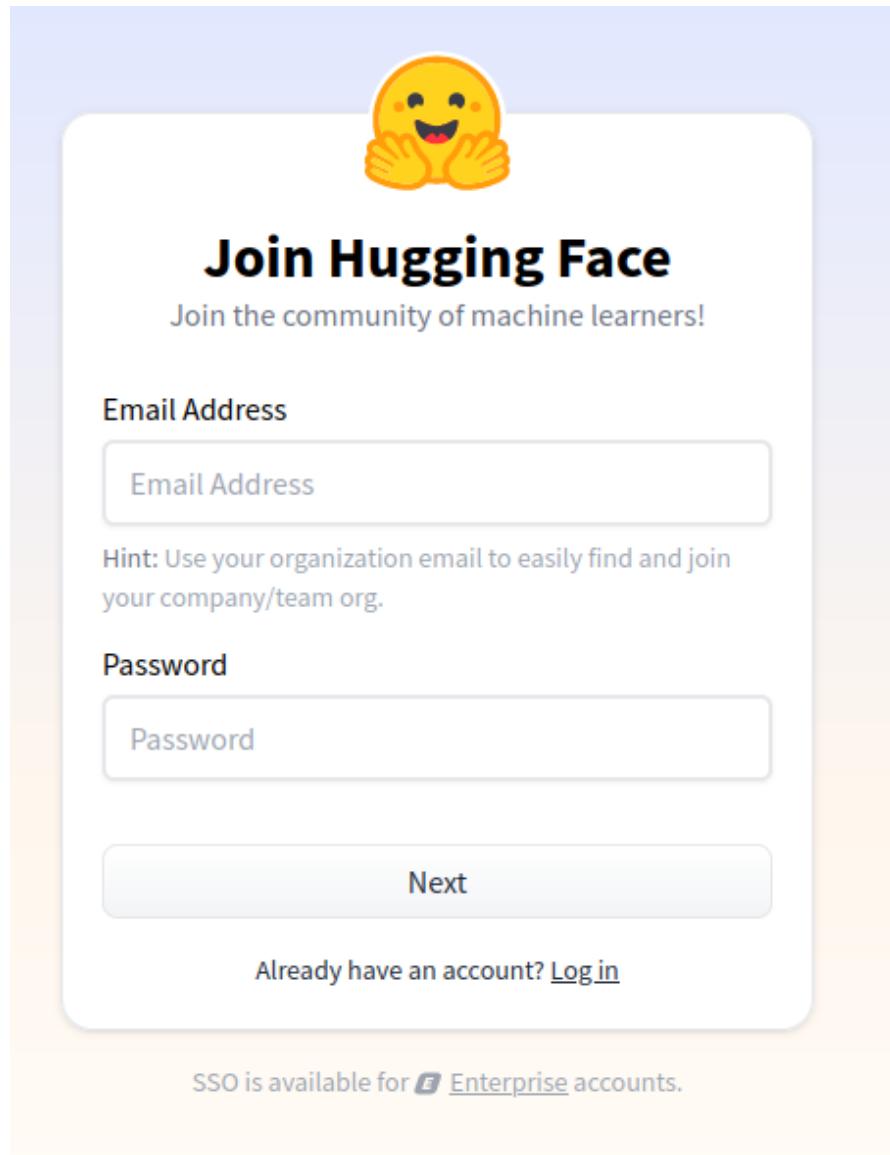


Figure 19: Criando uma conta no Hugging Face

Pedindo acesso a um modelo

De volta na página do modelo, você deverá ver um botão para solicitar acesso. O processo é diferente para cada modelo restrito. No caso do modelo do Google, basta ler e concordar com termos de uso.

Gerando um token de acesso

Vá para a página de gerenciamento de tokens, nas opções da plataforma ([link direto aqui](#)) e clique em New token. Dê um nome qualquer para seu token e mantenha a opção role como read.

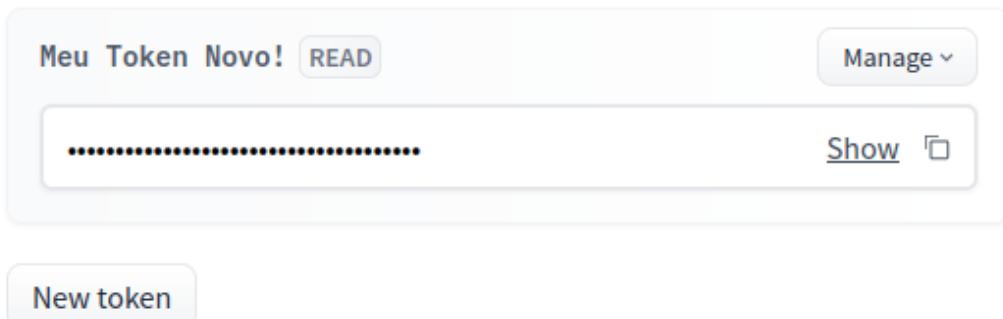


Figure 20: Gerando um token de acesso

Adicionando o token no código

Para adicionarmos o token no pipeline, simplesmente usamos o argumento `token` e passamos o token gerado na forma de string (substitua o 'xxxxx' pelo seu token de verdade):

```
from transformers import pipeline

chatbot = pipeline("text-generation", model="google/gemma-7b-it", token='xxxxx')
```

Com o token correto, o código deve começar a baixar o modelo.

Lembre-se de que este é um modelo pesado! Talvez seja melhor pausar o código acima e acessá-lo pela Inference API. Neste caso, o token é passado junto de um outro argumento de `requests.post` chamado `headers`. Use da seguinte forma:

```
import requests

modelo = 'google/gemma-7b-it'
url = f"https://api-inference.huggingface.co/models/{modelo}"

json = {
    'inputs': 'Olá, qual o seu nome?',
    'options': {'use_cache': False, 'wait_for_model': True},
}

token = 'xxxxx'
```

```
headers = {'Authorization': f'Bearer {token}'}

response = requests.post(url, json=json, headers=headers)
```

Melhores práticas ao usar o token: biblioteca dotenv

Em geral, não queremos escrever o token diretamente no código, como no exemplo acima. Isso porque, se compartilharmos o código ou disponibilizarmos publicamente (por exemplo, subindo ele no GitHub), o nosso token ficará exposto e qualquer pessoa pode se passar por nós no Hugging Face.

Uma solução comum para isso é ler o token a partir de um arquivo de texto. No caso de Python, existe a biblioteca `dotenv` que lê um arquivo de texto padrão (chamado `.env`) e passa os valores de lá para dentro das variáveis de ambiente. Então nosso código acessa as variáveis de ambiente diretamente, sem expor nossos tokens.

Como usar o arquivo `.env`

Vamos instalar a biblioteca `dotenv` com o comando abaixo:

```
pip install python-dotenv
```

Então, criamos um arquivo de texto chamado `.env` e colocamos o token lá:

```
HF_TOKEN='xxxxxx'
```

Atenção: em Mac/Linux, arquivos começados por ponto são ocultos por padrão.

Agora, modificamos nosso código para ler o token do arquivo `.env`. A função `dotenv.load_dotenv()` já procura por este arquivo por padrão e insere os valores dentro do dicionário de variáveis de ambiente, que acessamos com `os.environ`.

O código final fica:

```
import os

import dotenv
import requests

dotenv.load_dotenv()

modelo = 'google/gemma-7b-it'
url = f"https://api-inference.huggingface.co/models/{modelo}"

json = {
    'inputs': 'Olá, qual o seu nome?',
    'options': {'use_cache': False, 'wait_for_model': True},
}
```

```
token = os.environ['HF_TOKEN']
headers = {'Authorization': f'Bearer {token}'}

response = requests.post(url, json=json, headers=headers)
print(response.json())
```

Eis uma resposta do modelo (note que este modelo também foi treinado em português):

```
[{'generated_text': 'Olá, qual o seu nome?\n\nmeu nome é Carlos.\n\nOlá, Carlos.\n\nque\n→ quer que eu faça?'}]
```

Chegamos no mesmo “problema” das aulas anteriores: não estruturamos o template da pergunta em usuário e assistente, portanto o modelo está conversando consigo mesmo. Mas é só aplicar os conhecimentos das aulas anteriores para estruturar a resposta. De qualquer forma, conseguimos o que queríamos: acessar um modelo restrito com autenticação diretamente no código!

11. Miniprojeto - webapp com múltiplos chatbots

O código abaixo representa um pequeno webapp criado com Streamlit, que utiliza a Inference API para conversar com diferentes IAs. Assista à aula para entender o passo a passo, e utilize este projeto como ponto de partida para outras ideias!

```
import os

import dotenv
import requests
import streamlit as st
from transformers import AutoTokenizer

dotenv.load_dotenv()
token = os.environ['HF_TOKEN']

modelos = {
    'mistralai/Mixtral-8x7B-Instruct-v0.1': '[/INST]',
    'google/gemma-7b-it': '<start_of_turn>model\n',
}

nome_modelo = st.selectbox('Selecione um modelo:', options=modelos)
token_modelo = modelos[nome_modelo]

url = f"https://api-inference.huggingface.co/models/{nome_modelo}"
tokenizer = AutoTokenizer.from_pretrained(nome_modelo)

if (
    'modelo_atual' not in st.session_state # Primeira execução do programa
    or st.session_state['modelo_atual'] != nome_modelo # Outro modelo escolhido
):
    # Reiniciar o chat e redefinir o nome do modelo atual
    st.session_state['mensagens'] = []
    st.session_state['modelo_atual'] = nome_modelo

mensagens = st.session_state['mensagens']

area_chat = st.empty()
pergunta_usuario = st.chat_input('Faça sua pergunta aqui: ')

if pergunta_usuario:
    mensagens.append({'role': 'user', 'content': pergunta_usuario})
    inputs = tokenizer.apply_chat_template(mensagens, tokenize=False,
    ↪ add_generation_prompt=True)
    json = {
        'inputs': inputs,
        'parameters': {'max_new_tokens': 1_000},
        'options': {'use_cache': False, 'wait_for_model': True},
    }
    headers = {
        'Authentication': f'Bearer {token}',
    }
```

Explorando o Universo das IAs com Hugging Face

```
response = requests.post(url, json=json, headers=headers).json()
resposta_chatbot = response[0]['generated_text'].split(token_modelo)[-1]
mensagens.append({'role': 'assistant', 'content': resposta_chatbot})

with area_chat.container():
    for mensagem in mensagens:
        chat = st.chat_message(mensagem['role'])
        chat.markdown(mensagem['content'])
```

Aqui usamos *containers* próprios do Streamlit para estruturas de chat. Eis o resultado:

Selecione um modelo:

google/gemma-7b-it



Olá! você pode me ajudar com este problema de Python? Não sei como criar uma função que retorne a soma de 3 números :(



Função para retornar a soma de 3 números:

```
def soma_tres_numeros(num1, num2, num3):  
    return num1 + num2 + num3
```

Explicação:

- A função `soma_tres_numeros` recebe 3 argumentos: `num1`, `num2` e `num3`.
- A função retorna a soma dos 3 números com a expressão `num1 + num2 + num3`.

Exemplo de uso:

```
num1 = 10  
num2 = 20  
num3 = 30  
  
soma = soma_tres_numeros(num1, num2, num3)  
  
print(soma) # Saída: 60
```

Faça sua pergunta aqui:



Figure 21: Resultado do miniprojeto

12. Utilizando IAs de tradução

Os modelos de tradução funcionam de forma parecida com os outros que vimos até aqui. Um ponto importante é conferir **para quais línguas eles foram treinados**. Os maiores modelos foram treinados em múltiplas línguas, mas modelos menores podem excluir alguma língua relevante (no nosso caso, o português).

IA de tradução

Vamos testar o modelo [mBART-50 many to many](#) do Facebook. Note que precisamos passar as línguas de origem (`src_lang`) e destino (`dst_lang`) como argumentos:

```
from transformers import pipeline

modelo = "facebook/mbart-large-50-many-to-many-mmt"
mensagem = "Olá! Estou aprendendo a programar em Python e a usar modelos de inteligência
↪ artificial pelo Hugging Face."

tradutor = pipeline("translation", model=modelo)
traducao = tradutor(mensagem, src_lang='pt_XX', tgt_lang='en_XX')

print(traducao)
```

E o output:

```
[{'translation_text': "Hi! I'm learning to code in Python and using artificial intelligence
↪ models from the Hugging Face."}]
```

Parâmetros da tradução

Com quais línguas o modelo trabalha? Em geral, tanto as línguas quanto os seus respectivos códigos isto estarão explícitos no cartão do modelo:

⌚ Languages covered

Arabic (ar_AR), Czech (cs_CZ), German (de_DE), English (en_XX), Spanish (es_XX), Estonian (et_EE), Finnish (fi_FI), French (fr_XX), Gujarati (gu_IN), Hindi (hi_IN), Italian (it_IT), Japanese (ja_XX), Kazakh (kk_KZ), Korean (ko_KR), Lithuanian (lt_LT), Latvian (lv_LV), Burmese (my_MM), Nepali (ne_NP), Dutch (nl_XX), Romanian (ro_RO), Russian (ru_RU), Sinhala (si_LK), Turkish (tr_TR), Vietnamese (vi_VN), Chinese (zh_CN), Afrikaans (af_ZA), Azerbaijani (az_AZ), Bengali (bn_IN), Persian (fa_IR), Hebrew (he_IL), Croatian (hr_HR), Indonesian (id_ID), Georgian (ka_GE), Khmer (km_KH), Macedonian (mk_MK), Malayalam (ml_IN), Mongolian (mn_MN), Marathi (mr_IN), Polish (pl_PL), Pashto (ps_AF), Portuguese (pt_XX), Swedish (sv_SE), Swahili (sw_KE), Tamil (ta_IN), Telugu (te_IN), Thai (th_TH), Tagalog (tl_XX), Ukrainian (uk_UA), Urdu (ur_PK), Xhosa (xh_ZA), Galician (gl_ES), Slovene (sl_SI)

Figure 22: Línguas aceitas pelo modelo

Lembre-se também que os argumentos para o modelo nem sempre são `src_lang` e `dst_lang`. Isso varia com o modelo em si.

Outros testes

Podemos usar o modelo para traduzir diversas frases em sequência. Para isso, basta passarmos uma lista de mensagens, que o modelo colocará a tradução na lista de saída:

```
from transformers import pipeline

modelo = "facebook/mbart-large-50-many-to-many-mmt"
mensagens = [
    "Olá! Estou aprendendo a programar em Python e a usar modelos de inteligência artificial",
    "→ pelo Hugging Face.",",
    "Vamos nos encontrar às 15h da próxima sexta-feira. Eu acho que todos os meus amigos vão",
    "→ estar lá!",",
    "Três tigres tristes comeram três pratos de trigo.",",
    "Ser feliz sem motivo é a mais autêntica forma de felicidade.",",
]
linguas = [
    'en_XX',
    'es_XX',
    'fr_XX',
]

tradutor = pipeline("translation", model=modelo)
```

```
for lingua in linguas:
    print(f'Traduzindo do português para {lingua}...')
    traducoes = tradutor(mensagens, src_lang='pt_XX', tgt_lang=lingua)
    for mensagem, traducao in zip(mensagens, traducoes):
        print(f'Frase original: "{mensagem}"')
        frase_traduzida = traducao['translation_text']
        print(f'Frase em {lingua}: "{frase_traduzida}"')
```

E a saída:

```
Traduzindo do português para en_XX...
Frase original: "Olá! Estou aprendendo a programar em Python e a usar modelos de inteligência
→ artificial pelo Hugging Face."
Frase em en_XX: "Hi! I'm learning to code in Python and using artificial intelligence models
→ from the Hugging Face."
Frase original: "Vamos nos encontrar às 15h da próxima sexta-feira. Eu acho que todos os meus
→ amigos vão estar lá!"
Frase em en_XX: "We'll meet at 15 o'clock next Friday, and I think all my friends are going to
→ be there.

[ ... ]
```

Ou se preferirmos, podemos fazer este mesmo processo pela Inference API (desde que a API esteja ativada para o modelo, é claro):

```
import requests

modelo = "facebook/mbart-large-50-many-to-many-mmt"
url = f"https://api-inference.huggingface.co/models/{modelo}"

mensagens = [
    "Olá! Estou aprendendo a programar em Python e a usar modelos de inteligência artificial
    → pelo Hugging Face.",
    "Vamos nos encontrar às 15h da próxima sexta-feira. Eu acho que todos os meus amigos vão
    → estar lá!",
    "Três tigres tristes comeram três pratos de trigo.",
    "Ser feliz sem motivo é a mais autêntica forma de felicidade.",
]
linguas = [
    'en_XX',
    'es_XX',
    'fr_XX',
]

for lingua in linguas:
    print(f'Traduzindo do português para {lingua}...')
    json = {
        'inputs': mensagens,
        'parameters': {'src_lang': 'pt_XX', 'tgt_lang': lingua},
        'options': {'use_cache': False, 'wait_for_model': True},
    }
    response = requests.post(url, json=json)
    traducoes = response.json()
    for mensagem, traducao in zip(mensagens, traducoes):
```

```
print(f'Frase original: "{mensagem}"')
frase_traduzida = traducao['translation_text']
print(f'Frase em {lingua}: "{frase_traduzida}"')
```

Para quê usar modelos de tradução?

Perto dos outros modelos, um modelo de tradução não parecem tão “úteis”. Afinal, já estamos acostumados com ferramentas como Google Translate muito antes de falarmos de IAs!

Contudo, estes modelos possuem aplicação prática sim. Nas aulas anteriores, percebemos que as melhores IAs tendem a funcionar apenas em inglês (infelizmente). Mesmo nesses casos, ainda é possível usá-las se adicionarmos um modelo de tradução a uma de suas funcionalidades. Alguns exemplos:

- Resumir um texto em português fazendo tradução para inglês -> modelo de resumo (em inglês) -> tradução do resumo para português
- Transformar áudio em texto (em inglês) e traduzi-lo para português -> modelo que traduz áudios em inglês para textos em português

Assim, conseguimos adaptar e juntar IAs diferentes no nossos projetos!

É claro que adicionarmos uma IA a mais pode trazer maior variabilidade dos resultados. Mas por outro lado, faz total sentido usarmos IAs especializadas para tarefas específicas e apenas ir “ligando os pontos”, ao invés de tentarmos desenvolver um modelo gigantesco capaz de fazer tudo ao mesmo tempo.

13. Utilizando IAs de resumo

Os modelos de resumo funcionam de forma parecida com os demais que vimos até aqui. Devemos apenas prestar atenção nas línguas com que eles foram treinados. Nesta aula, vamos testar alguns modelos de resumo usando a Inference API - afinal, eles são modelos grandes!

Resumindo textos em português

Vamos utilizar [este modelo](#) (treinado para múltiplas línguas) para resumir alguns parágrafos de texto em português.

Resumindo um artigo da Wikipedia

Vamos tentar resumir os primeiros parágrafos do [artigo do Brasil na Wikipedia](#)(arquivo brasil.txt do material de aula):

Brasil, oficialmente República Federativa do Brasil, é o maior país da América do Sul e da
→ região da América Latina, sendo o quinto maior do mundo em área territorial (equivalente a
→ 47,3% do território sul-americano), com 8 510 417,771 km², e o sétimo em população (com
→ 203 milhões de habitantes, em agosto de 2022). É o único país na América onde se fala
→ majoritariamente a língua portuguesa e o maior país lusófono do planeta, além de ser uma
→ das nações mais multiculturais e etnicamente diversas, em decorrência da forte imigração
→ oriunda de variados locais do mundo. Sua atual Constituição, promulgada em 1988, concebe o
→ Brasil como uma república federativa presidencialista, formada pela união dos 26 estados,
→ do Distrito Federal e dos 5 571 municípios.

Banhado pelo Oceano Atlântico, o Brasil tem um litoral de 7 491 km e faz fronteira com todos
→ os outros países sul-americanos, exceto Chile e Equador, sendo limitado a norte pela
→ Venezuela, Guiana, Suriname e pelo departamento ultramarino francês da Guiana Francesa; a
→ noroeste pela Colômbia; a oeste pela Bolívia e Peru; a sudoeste pela Argentina e Paraguai
→ e ao sul pelo Uruguai. Vários arquipélagos formam parte do território brasileiro, como o
→ Atol das Rocas, o Arquipélago de São Pedro e São Paulo, Fernando de Noronha (o único
→ destes habitado por civis) e Trindade e Martim Vaz. O Brasil também é o lar de uma
→ diversidade de animais selvagens, ecossistemas e de vastos recursos naturais em uma grande
→ variedade de habitats protegidos.

Explorando o Universo das IAs com Hugging Face

O território que atualmente forma o Brasil foi oficialmente descoberto pelos portugueses em 22 de abril de 1500, em expedição liderada por Pedro Álvares Cabral. Segundo alguns historiadores como Antonio de Herrera e Pietro d'Anghiera, o encontro do território teria sido três meses antes, em 26 de janeiro, pelo navegador espanhol Vicente Yáñez Pinzón, durante uma expedição sob seu comando. A região, então habitada por indígenas ameríndios divididos entre milhares de grupos étnicos e linguísticos diferentes, cabia a Portugal pelo Tratado de Tordesilhas, e tornou-se uma colônia do Império Português. O vínculo colonial foi rompido, de fato, quando em 1808 a capital do reino foi transferida de Lisboa para a cidade do Rio de Janeiro, depois de tropas francesas comandadas por Napoleão Bonaparte invadirem o território português. Em 1815, o Brasil se torna parte de um reino unido com Portugal. Dom Pedro I, o primeiro imperador, proclamou a independência política do país em 1822. Inicialmente independente como um império, período no qual foi uma monarquia constitucional parlamentarista, o Brasil tornou-se uma república em 1889, em razão de um golpe militar chefiado pelo marechal Deodoro da Fonseca (o primeiro presidente), embora uma legislatura bicameral, agora chamada de Congresso Nacional, já existisse desde a ratificação da primeira Constituição, em 1824. Desde o início do período republicano, a governança democrática foi interrompida por longos períodos de regimes autoritários, até um governo civil e eleito democraticamente assumir o poder em 1985, com o fim da ditadura militar.

Como potência regional e média, a nação tem reconhecimento e influência internacional, sendo que também é classificada como uma potência global emergente e como uma potencial superpotência por vários analistas. O PIB nominal brasileiro é o nono maior do mundo e o oitavo por paridade do poder de compra (PPC), sendo, em ambos, o maior da América Latina e do Hemisfério Sul. O país é um dos principais celeiros do planeta, sendo o maior produtor de café dos últimos 150 anos, além de ser classificado como uma economia de renda média-alta pelo Banco Mundial e como um país recentemente industrializado, que detém a maior parcela de riqueza global da América do Sul. No entanto, o país ainda mantém níveis notáveis de corrupção, criminalidade e desigualdade social. É membro fundador da Organização das Nações Unidas (ONU), G20, BRICS, Comunidade dos Países de Língua Portuguesa (CPLP), União Latina, Organização dos Estados Americanos (OEA), Organização dos Estados Ibero-americanos (OEI) e Mercado Comum do Sul (Mercosul).

O código para uso com a Inference API fica:

```
import requests

modelo = "csebuetnlp/mT5_multilingual_XLSum"
url = f"https://api-inference.huggingface.co/models/{modelo}"

with open('brasil.txt') as f:
    texto = f.read()

json = {
    'inputs': texto,
    'options': {'use_cache': False, 'wait_for_model': True},
}
response = requests.post(url, json=json)
print(response.json())
```

Com isso, temos a saída:

```
[{'summary_text': 'O Brasil é o maior país da América do Sul e a única potência global emergente.'}]
```

Podemos controlar o número mínimo de tokens da resposta com o parâmetro `min_length` (e de forma análoga, o número máximo com `max_length`).

```
import requests

modelo = "csebuetnlp/mT5_multilingual_XLSum"
url = f"https://api-inference.huggingface.co/models/{modelo}"

with open('brasil.txt') as f:
    texto = f.read()

json = {
    'inputs': texto,
    'parameters': {'min_length': 200}, # Em tokens
    'options': {'use_cache': False, 'wait_for_model': True},
}
response = requests.post(url, json=json)
print(response.json())
```

Agora, temos a saída:

```
[{'summary_text': 'O Brasil é um dos países mais ricos do mundo, com o maior número de habitantes em área territorial e de população. Além de ser uma potência global emergente, ela também tem níveis notáveis de corrupção, criminalidade, desigualdade social, pobreza, violência, crimes contra a humanidade'}]
```

Resumindo uma notícia

Vamos tentar resumir os primeiros parágrafos [dessa notícia](#)(arquivo `noticia.txt`):

Câmara dos EUA aprova projeto de lei que pode banir TikTok no país

A Câmara dos Deputados dos EUA aprovou nesta quarta-feira (13) uma lei que pode banir o TikTok no país. O projeto ordena que o app, controlado pela empresa chinesa ByteDance, tenha um novo dono nos EUA. Caso contrário, a rede social terá que deixar o país.

Com amplo apoio de democratas e republicanos, o projeto de lei foi aprovado por 352 votos a 65. Para valer, ele ainda precisa passar pelo Senado e também depende de sanção do presidente dos EUA. Na semana passada, o presidente Joe Biden havia dito que assinaria a lei.

Por que a lei foi proposta? Apesar de ser focada no TikTok, a legislação proposta afeta "aplicativos controlados por adversários estrangeiros" (entenda abaixo).

A ideia vem desde o governo de Donald Trump, que dizia que a ByteDance representava um risco para a segurança do país porque a China poderia se aproveitar do poder da empresa para obter dados de usuários americanos. A controladora do TikTok nega esse risco.

Segundo a agência Reuters, o TikTok disse que espera que "o Senado americano considere os fatos e que escute seus eleitores" antes de tomar qualquer decisão.

"Esperamos que eles percebam o impacto na economia, em 7 milhões de pequenas empresas e nos → 170 milhões de americanos que usam nosso serviço", disse um porta-voz do TikTok após o → resultado.

O código fica essencialmente o mesmo, basta apenas mudar o arquivo de origem de `brasil.txt` para `noticia.txt`.

Eis o resumo quando não passamos um limite de tokens:

```
[{'summary_text': 'O app TikTok está sendo alvo de uma polêmica.'}]
```

E com `{"min_length": 100}`:

```
[{'summary_text': 'O app TikTok é um dos mais populares aplicativos do mundo, mas o governo → dos Estados Unidos quer que ele tenha um novo dono nos EUA e não seja alvo de sanções do → presidente Joe Biden, que está sendo acusado de envolvimento no serviço com os seus → usuários estrangeiros, segundo'}]
```

Outros parâmetros

Existem outros parâmetros possíveis de usarmos nestes modelos. É possível ajustar a temperatura, valor de `top_k` e outros valores. Você pode conferir detalhes neste [link da documentação da Inference API](#).

14. Classificando textos com IAs

Classificação de texto (análise de sentimento)

A tarefa de classificação de texto é também conhecida com análise de sentimento. A ideia básica é a seguinte: dado um trecho de um texto, qual sentimento ou emoção ele representa?

Vamos ver na sequência alguns modelos que trabalham de diferentes formas com análise de sentimentos.

Classificando reviews de um produto

O modelo [distilbert](#) é capaz de avaliar se uma frase apresenta um sentimento positivo, negativo, ou neutro, em múltiplas línguas.

O código abaixo usa esse modelo para classificar frases de reviews de produtos (as frases foram extraídas de produtos reais no Mercado Livre):

```
import requests
from transformers import pipeline

modelo = "lxyuan/distilbert-base-multilingual-cased-sentiments-student"
classificador = pipeline("text-classification", model=modelo)

reviews = [
    "Até então não tenho do que reclamar. Estou usando pra estudo, está bem tranquilo até
     → aqui.",
    "Acho que vale o custo benefício, caso seja para usos básicos.",
    "A bateria do notebook está descarregando muito rápido.",
    "Eu estava com muito medo de me arrepender da compra. Mas eu realmente gostei! Ótimo
     → demais, comprem!",
    "Muito bom, recomendo!",
    "Não comprem, caro demais pelo que oferece.",
    "Super custo benefício, pelo preço que paguei superou todas as minhas expectativas.",
    "Excelente, zero arrependimentos. Muito muito bom.",
    "Esperava um pouco mais. Mas é um produto bom. Não coloquei mais estrelas pois não usei
     → direito.",
]

for review in reviews:
    print(f'Avaliação: "{review}"')
    resultado = classificador(review)
    prob = resultado[0]['score'] * 100
    print('Análise de sentimento:', resultado[0]['label'], f'{prob:.2f}%')
```

O output fica algo como:

```
Avaliação: "Até então não tenho do que reclamar. Estou usando pra estudo, está bem tranquilo
     → até aqui."
Análise de sentimento: negative 40.40%
```

Avaliação: "Acho que vale o custo benefício, caso seja para usos básicos."
Análise de sentimento: positive 84.37%

[...]

O modelo não acerta todos os casos, mas parece acertar a maioria deles.

Note que este modelo não é especializado em reviews, nem em língua portuguesa, então ser capaz de fazer uma análise relativamente precisa como esta é surpreendente!

Neste caso, o modelo retornou apenas o sentimento de maior probabilidade. Se quisermos que ele retorne todas as opções, podemos adicionar o argumento top_k=None:

```
from transformers import pipeline

modelo = "lxyuan/distilbert-base-multilingual-cased-sentiments-student"
classificador = pipeline("text-classification", model=modelo)

review = "Acho que vale o custo benefício, caso seja para usos básicos."
resultado = classificador(review, top_k=None)

print(review)
print('Análise de sentimento:', resultado)
```

Outros exemplos de classificação

Vamos passar agora por uma série de exemplos de modelos, com diferentes funcionalidades de classificação.

Classificando emoções

Modelo: Roberta-base go_emotions:

```
from transformers import pipeline

classificador = pipeline(task="text-classification", model="SamLowe/roberta-base-go_emotions")

frases = [
    'I am feeling a bit better, thanks for asking!',
    'Well, it could be worse.',
    'This is awful!',
    'I am very happy with the test results!',
    'Unbelievable!',
    'Go away!!!!',
]
for frase in frases:
    print(f'frase: "{frase}"')
    resultado = classificador(frase)
```

```
prob = resultado[0]['score'] * 100
print('Análise de sentimento:', resultado[0]['label'], f'{prob:.2f}%')
```

Veja que o output deste modelo não é apenas positivo ou negativo, mas o nome da emoção detectada!

Classificando headlines de frases relacionadas a finanças

Modelo: [Finbert](#):

```
import requests

url = "https://api-inference.huggingface.co/models/ProsusAI/finbert"
frases = [
    'However, the growth margin slowed down due to the financial crisis.',
    'The company laid off thousands of employees last week.',
    'According to their updated strategy for the years 2009–2012, the company targets a
     ↳ long-term net sales growth',
    'Result before taxes decreased to nearly EUR 14.5mn, compared to nearly EUR 20mn in the
     ↳ previous accounting period.'
]

for frase in frases:
    json = {
        'inputs': frase,
        'options': {'use_cache': False, 'wait_for_model': True},
    }
    response = requests.post(url, json=json)
    print('Frase original:', frase)
    print(response.json(), '\n')
```

Classificando ironia em tweets

Modelo: [twitter-Roberta-base irony](#):

```
from transformers import pipeline

detector_ironia = pipeline("text-classification",
                           model="cardiffnlp/twitter-roberta-base-irony", top_k=None)

for frase in [
    "@Mountgrace lol i know! its so frustrating isnt it?!", 
    "I don't like clowns but I'm going to be one.", 
    "Now I remember why I buy books online @user #servicewithasmile", 
    "Simply having a wonderful christmas time :D",
]:
    print(f'frase: "{frase}"')
    resultado = detector_ironia(frase)
    resultado_mais_provavel = resultado[0][0]
```

```
prob = resultado_mais_provavel['score'] * 100
print('Análise de sentimento:', resultado_mais_provavel['label'], f'{prob:.2f}%')
```

Classificação sem contexto (*zero-shot*)

Em todos os casos acima, os modelos de IA foram treinados para classificar textos em algum dos valores de resposta conhecidos. Ou seja, o modelo sabia de antemão quais eram as possíveis respostas para cada tipo de tarefa, como “irônico” e “não-irônico” no caso dos tweets.

Na classificação sem contexto, o modelo não sabe de antemão que classificadores usar. Somos nós que informamos ao modelo os nomes das classes possíveis, e ele tem que “se virar” para tentar entender qual a classificação mais provável para um dado texto.

Estes são os chamados modelos *zero-shot*, pois eles têm zero exemplos de como desempenhar sua tarefa.

Este é o resultado do modelo [bart-large-mnli](#), testado através da Inference API:

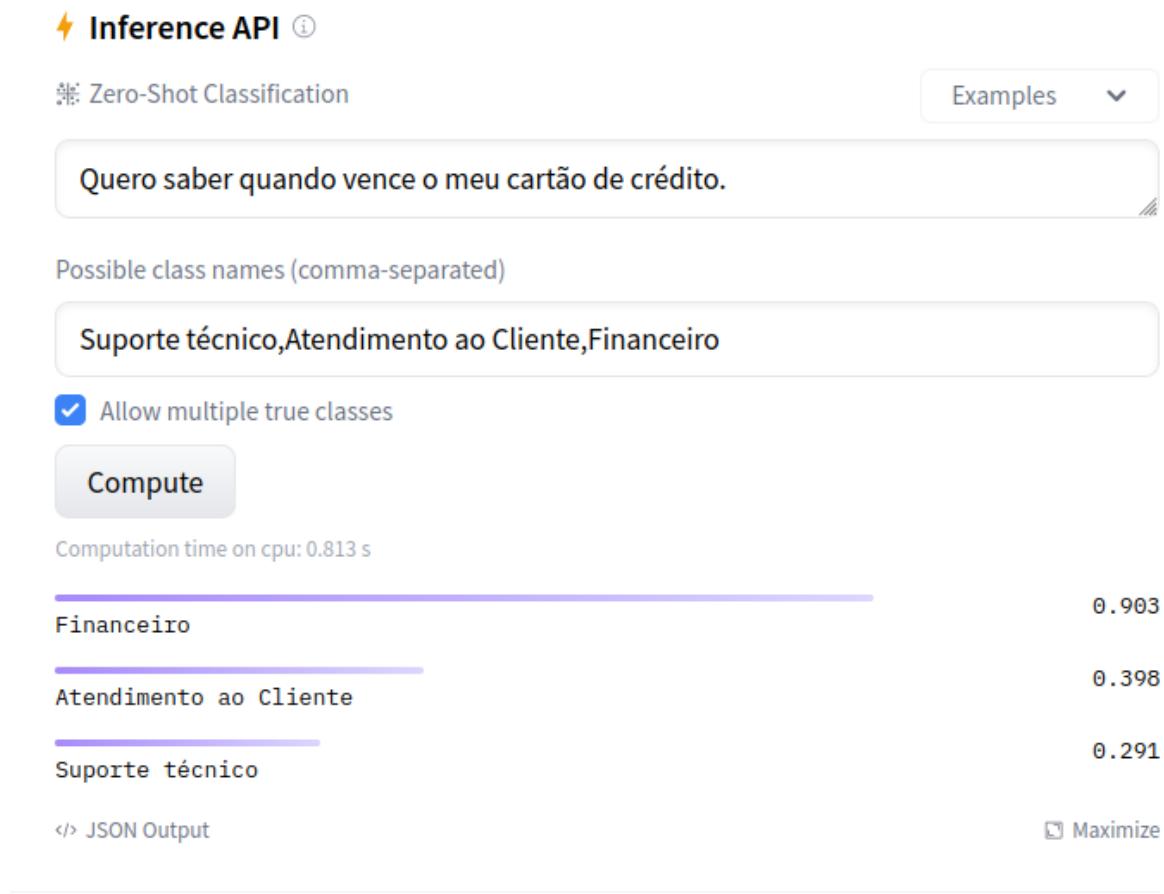


Figure 23: Teste de um modelo de classificação zero-shot.

Vale a pena usar um modelo zero-shot?

Modelos de classificação zero-shot são interessantes em teoria. Na prática, é difícil pensar em um uso onde eles se sobressairiam a um modelo de IA que classifique de acordo com alguma tarefa específica.

Sua vantagem é que um único modelo de IA seria capaz de atuar em qualquer tipo de classificação. Contudo, como já vimos ao longo deste curso, muitas vezes faz mais sentido termos modelos que atuam de forma específica em um problema único.

Fica aqui o convite para testar os modelos zero-shot de classificação do Hugging Face e procurar por aplicações práticas deles!

15. Acessando Datasets do Hugging Face

Como vimos no começo do curso, o Hugging Face também possui conjuntos de dados (datasets), além dos modelos de IA. Vamos aprender como interagir com estes Datasets tanto pela interface do Hugging Face quanto por Python.

A anatomia de um Dataset

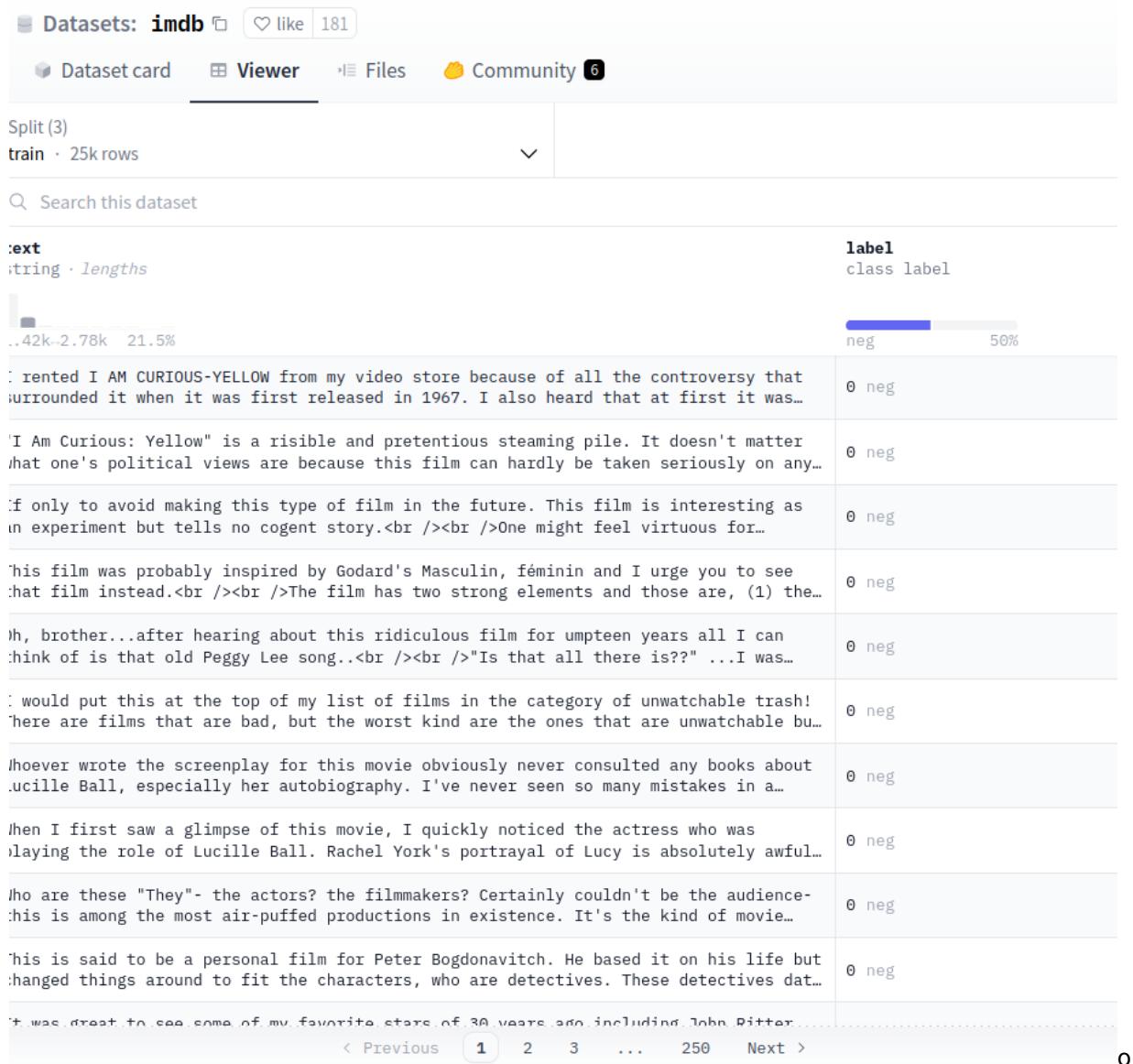
Vamos selecionar um Dataset para a tarefa `text-classification` e com bastante número de downloads.

The screenshot shows the Hugging Face Datasets interface. At the top, there's a search bar and a navigation menu with links for Models, Datasets, Spaces, Posts, Docs, Solutions, and Pricing. Below the menu, a sub-navigation bar includes Tasks, Sizes, Sub-tasks, Languages, Licenses, and Other. A sidebar on the left lists categories like Multimodal, Computer Vision, and Natural Language Processing, with 'Text Classification' highlighted. The main area displays a list of datasets with their names, last updated date, size, and download count. The 'imdb' dataset is highlighted with a red box. The sorting dropdown at the top right is set to 'Most downloads'. The 'imdb' dataset card shows it was last updated on Jan 4, has 385k files, and 181 downloads.

Figure 24: Escolha do Dataset.

O Dataset é formado por [reviews de filmes do IMDB](#), junto de uma classe (0 ou 1) que determina se o review é positivo ou negativo. Acesse a aba de Dataset Viewer para vê-lo em mais detalhes:

Explorando o Universo das IAs com Hugging Face



O Dataset em si está na aba `Files` and `versions`, no formato `.parquet`. Poderíamos baixá-lo diretamente pela interface, mas vamos fazer isso por Python.

Acessando Datasets por Python

O Hugging Face possui uma biblioteca específica para acessar seus Datasets chamada (de forma não muito criativa) de `datasets`. Vamos instalá-la com:

```
pip install datasets
```

Voltando para nosso código, podemos acessar o Dataset com:

```
from datasets import load_dataset  
  
dataset = load_dataset("imdb")
```

Mais uma vez, o Hugging Face simplifica nossa vida! O Dataset será baixada para a mesma pasta de cache que o Hugging Face usa para os modelos.

O que há no Dataset?

Vamos printar a variável e ver o que ela contém:

```
print(dataset)  
  
# output:  
# DatasetDict({  
#     train: Dataset({  
#         features: ['text', 'label'],  
#         num_rows: 25000  
#     })  
#     test: Dataset({  
#         features: ['text', 'label'],  
#         num_rows: 25000  
#     })  
#     unsupervised: Dataset({  
#         features: ['text', 'label'],  
#         num_rows: 50000  
#     })  
# })
```

Um Dataset vem no formato de `DatasetDict`. Só pelo nome podemos esperar que seja um objeto parecido com um dicionário de Python.

Splits

Vemos também que os dados estão divididos em splits. Quando pensamos em treinar uma IA, geralmente temos que dividir os dados em dados de treino (que vamos passar para o modelo “aprender” a sua função) e teste (que vamos usar para testar a performance do modelo após o treino).

No caso desse Dataset, há ainda um split de dados “unsupervised”, que são dados não anotados. Os demais splits dos dados (“train” e “test”) foram classificados manualmente por algum avaliador humano (ou por um conjunto deles). Em geral, Datasets de boa qualidade irão detalhar este processo de anotação lá no seu card.

Linhas e colunas

O Dataset também indica o nome das colunas (chamadas aqui de `features`) e número de linhas (`num_rows`) de cada split.

Acessando dados

Podemos usar a mesma sintaxe de dicionário para acessar um split do Dataset:

```
dataset_treino = dataset['train']
print(dataset_treino)

# output:
# Dataset({
#     features: ['text', 'label'],
#     num_rows: 25000
# })
```

Tendo acessado um split do Dataset, podemos então acessar um de seus dados usando o índice da linha. Por exemplo, para acessar os dados da linha 10 do Dataset de treino, usamos o código a seguir:

```
# Lembrando que o índice começa de zero!
print(dataset_treino[9])

# output:
# {'text': 'It was great to see some of my favorite stars of 30 years ago including John
# Ritter, Ben Gazarra and Audrey Hepburn. They looked quite wonderful. But that was it. They
# were not given any characters or good lines to work with. I neither understood or cared
# what the characters were doing.<br /><br />Some of the smaller female roles were fine,
# Patty Henson and Colleen Camp were quite competent and confident in their small sidekick
# parts. They showed some talent and it is sad they didn\'t go on to star in more and better
# films. Sadly, I didn\'t think Dorothy Stratton got a chance to act in this her only
# important film role.<br /><br />The film appears to have some fans, and I was very
# open-minded when I started watching it. I am a big Peter Bogdanovich fan and I enjoyed his
# last movie, "Cat\'s Meow" and all his early ones from "Targets" to "Nickleodeon". So, it
# really surprised me that I was barely able to keep awake watching this one.<br /><br />It
# is ironic that this movie is about a detective agency where the detectives and clients get
# romantically involved with each other. Five years later, Bogdanovich\'s ex-girlfriend,
# Cybil Shepherd had a hit television series called "Moonlighting" stealing the story idea
# from Bogdanovich. Of course, there was a great difference in that the series relied on
# tons of witty dialogue, while this tries to make do with slapstick and a few screwball
# lines.<br /><br />Bottom line: It ain\'t no "Paper Moon" and only a very pale version of
# "What\'s Up, Doc".', 'label': 0}
```

O output do código é um dicionário, onde cada chave é o nome da coluna, e o valor respectivo é o valor daquela coluna e linha. Ou seja, se quisermos um valor específico de uma linha e coluna, podemos fazer algo como:

```
print(dataset_treino[9]['label'])
```

```
# output:  
# 0
```

Nesse caso, é um filme cujo review foi anotado como sendo negativo (valor zero).

Por outro lado, se quisermos todos os valores de uma coluna, podemos passar o nome da coluna diretamente para o Dataset. Isso retorna uma lista dos valores da coluna:

```
print(dataset_treino['label'])  
  
# output  
# [ 0, 0, 0, 0, 0, 0, 0, ...]
```

Convertendo um Dataset

Os Datasets nos permitem trabalhar com bibliotecas “tradicionais” de manipulação de dados em Python, como pandas. Para isso, basta usar o método `to_pandas()`:

```
df = dataset_treino.to_pandas()  
print(df)  
  
# output:  
# text  label  
# 0      I rented I AM CURIOUS-YELLOW from my video sto...  0  
# 1      "I Am Curious: Yellow" is a risible and preten...  0  
# 2      If only to avoid making this type of film in t...  0  
# 3      This film was probably inspired by Godard's Ma...  0  
# 4      Oh, brother...after hearing about this ridicul...  0  
# ...          ...  ...  
# 24995  A hit at the time but now better categorised a...  1  
# 24996  I love this movie like no other. Another time ...  1  
# 24997  This film and it's sequel Barry McKenzie holds...  1  
# 24998  'The Adventures Of Barry McKenzie' started lif...  1  
# 24999  The story centers around Barry McKenzie who mu...  1  
#  
# [25000 rows x 2 columns]
```

Agora podemos usar todos os métodos tradicionais de análise de dados que já conhecemos do pandas, ou então fazer a conversão para um tipo de arquivo suportado pelo pandas: Excel, CSV, Parquet, banco de dados SQL,...

Lidando com Datasets grandes

Este conjunto de dados é relativamente pequeno (< 100 MB de espaço em disco), mas não é difícil encontrar Datasets maiores que não sirvam na memória do computador. Neste caso, podemos adicionar o argumento `streaming=True` para carregar os dados em modo “Streaming”.

```
from datasets import load_dataset

dataset = load_dataset("imdb", streaming=True)
print(dataset)

# output:
# IterableDatasetDict({
#     train:Iterable Dataset({
#         features: ['text', 'label'],
#         n_shards: 1
#     })
#     test: IterableDataset({
#         features: ['text', 'label'],
#         n_shards: 1
#     })
#     unsupervised: IterableDataset({
#         features: ['text', 'label'],
#         n_shards: 1
#     })
# })
```

Veja que agora os objetos são do tipo `Iterable`. Isso significa que podemos iterar sobre estes objetos linha a linha, sem ocupar todo o espaço em memória:

```
dataset_treino = dataset['train']

for linha in dataset_treino:
    print(linha) # Primeira linha
    break
```

Também podemos usar o método `take` para pegar as N primeiras linhas do conjunto de dados:

```
linhas = dataset_treino.take(3)

print(linhas)
print(list(linhas))
```

Note que a variável `linhas` é um novo `IterableDataset`. Podemos usar a função `list()` para acessar todos os seus dados (como uma lista de dicionários).

Usar um Dataset iterável é consideravelmente mais lento, porém permite que consigamos iterar sobre todos os dados, não importa o quão grande é o Dataset!

16. Conhecendo a comunidade do Hugging Face

Como já falamos anteriormente, o Hugging Face tem o tamanho que tem principalmente por causa de sua **comunidade**. Vamos explorar alguns pontos relacionados a ela nesta aula.

Página inicial

Quando você abrir a página inicial do Hugging Face enquanto está logado, verá diversas opções relacionadas à comunidade. Você receberá sugestões de perfis ou organizações para seguir, e pode acompanhar os “posts” (atualizações) de cada perfil.

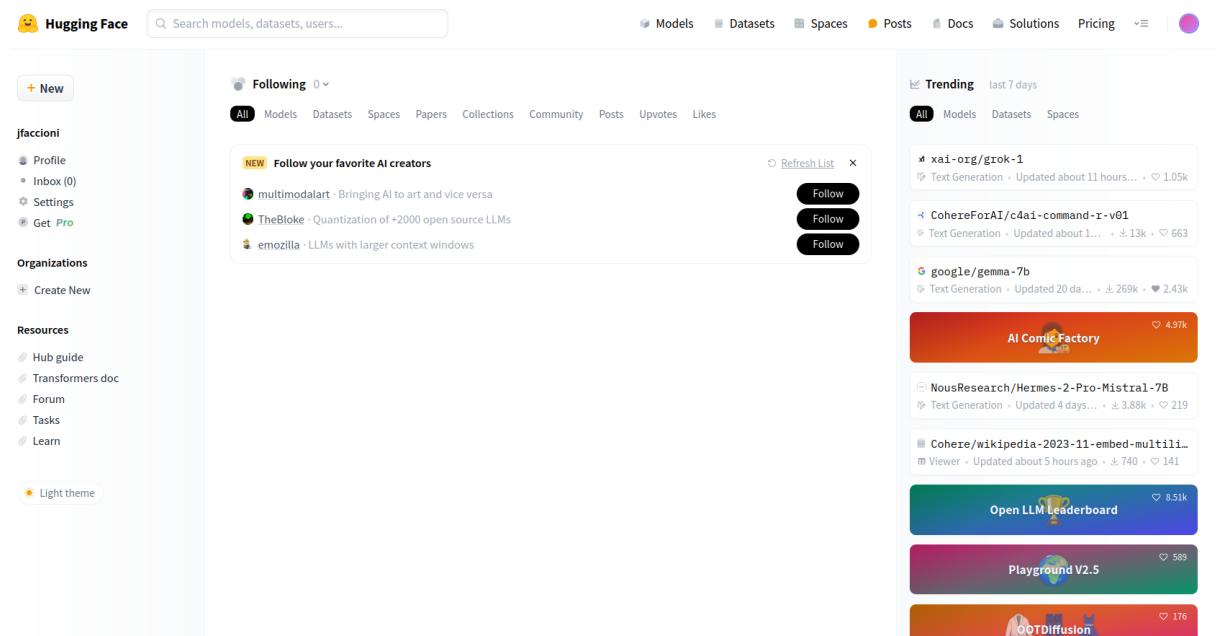


Figure 25: Página inicial do Hugging Face quando você está logado.

Opções de comunidade

No menu à direita, é possível selecionar diversas opções de comunidade, como artigos de blog, páginas de cursos do próprio Hugging Face, e acessos ao servidor de Discord, ao Forum e ao GitHub oficial.

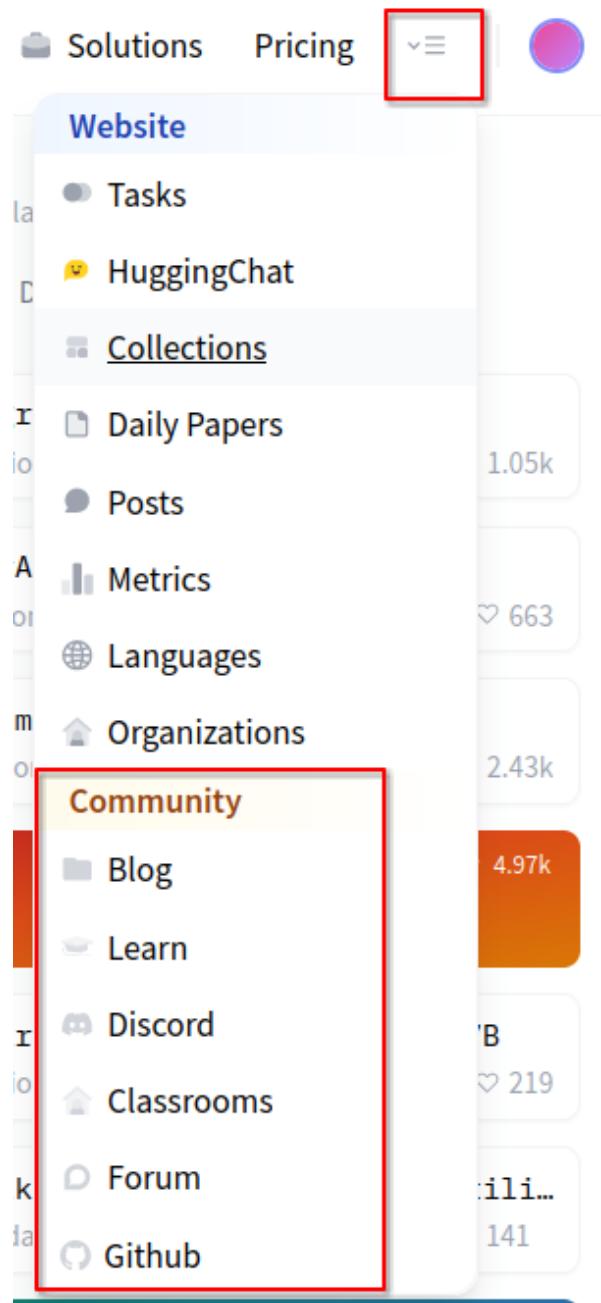


Figure 26: Menu de opções de comunidade.

Spaces

Os Spaces é onde a comunidade apresenta suas principais criações. É possível subir um aplicativo de IA forma bastante rápida utilizando os Spaces! Atualmente, **a criação de Spaces não é cobrada** pelo

Explorando o Universo das IAs com Hugging Face

Hugging Face, desde que você utilize a configuração mais básica.

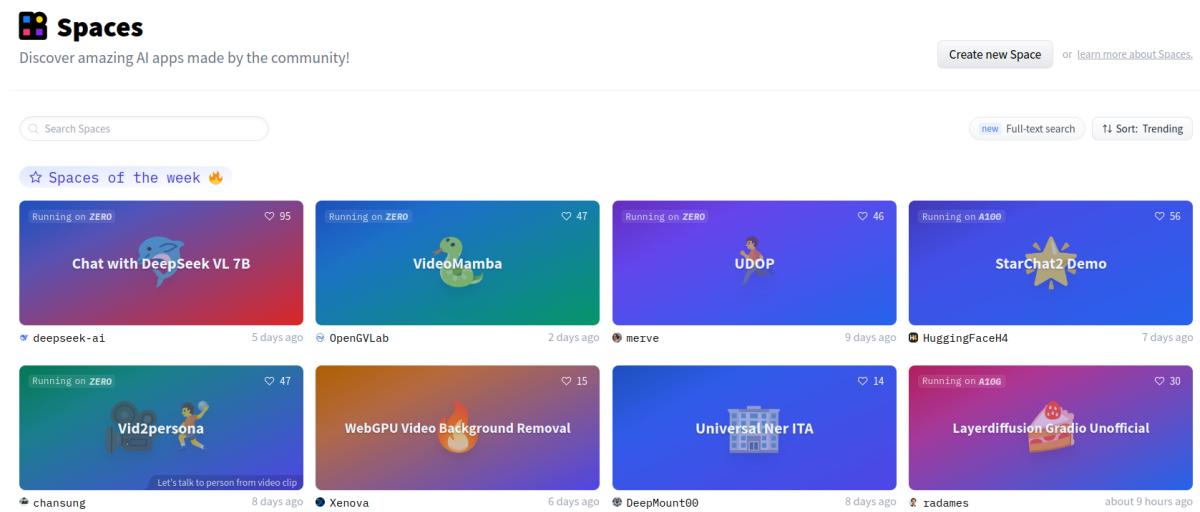


Figure 27: Página inicial dos Spaces do Hugging Face

Este [link](#) apresenta os principais features dos Spaces do Hugging Face.

17. Criando um Space com nosso webapp

Vamos aprender a subir nosso miniprojeto para um Space do Hugging Face!

Passo 1: criar um Space

Na página dos Spaces, Clique no botão Create new Space:

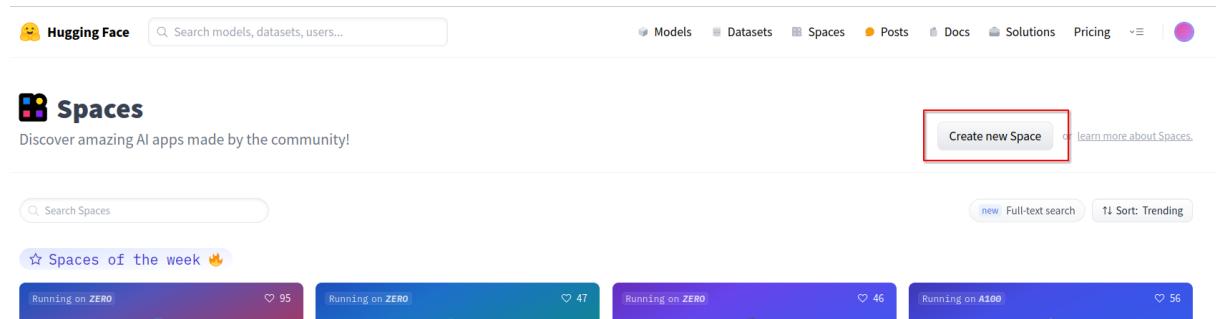


Figure 28: Botão para criar um novo Space.

Na tela a seguir, configure seu Space conforme as opções abaixo:

The screenshot shows a 'Create a new Space' form. At the top is a logo consisting of four colored squares (blue, yellow, red, green) arranged in a 2x2 grid. Below it is the title 'Create a new Space'. A descriptive text follows: 'Spaces are Git repositories that host application code for Machine Learning demos. You can build Spaces with Python libraries like Streamlit or Gradio, or using Docker images.' The form fields include:

- Owner:** A dropdown menu showing 'jfaccioni'.
- Space name:** An input field containing 'meu_space_teste'.
- License:** An input field containing 'mit'.
- Select the Space SDK:** A section with four options: Streamlit (red crown icon), Gradio (orange hexagon icon), Docker (blue ship icon with 'NEW' badge), and Static (blue square icon with 'NEW' badge). The Docker and Static options have '13 templates' and '3 templates' respectively listed below them.
- Space hardware:** A section showing 'CPU basic · 2 vCPU · 16 GB · FREE' with a 'Free' badge. A note below states: 'You can switch to a different hardware at any time in your Space settings. You will be billed for every minute of uptime on a paid hardware.'

Figure 29: Opções de configuração do Space sendo criado.

- **Space name:** Escolha um nome qualquer para seu Space. Este nome será parte da URL final do seu Space.
- **License:** Escolha a licença do seu Space. Se não souber qual escolher, MIT é um bom padrão, pois permite reprodução irrestrita e sem garantias.

- **Space SDK:** Aqui você escolhe com que ferramenta criar seu Space. O Hugging Face tem um suporte diferenciado para Streamlit e Gradio, mas é possível usar um ambiente qualquer com as opções de Docker e Static. Como criamos nosso app com Streamlit, escolha esta opção.
- **Space Hardware:** A configuração da máquina em que o Space vai rodar. **Mantenha a opção padrão para o Space ser gratuito!**

Você também terá de escolher se o Space vai ser público ou privado. Vamos manter público por enquanto (você pode alterar isso a qualquer momento).

Passo 2: adicionar seu código ao Space

Seu Space já está no ar! Porém, ele ainda não possui código. A tela abaixo mostra a cara do seu Space no momento:

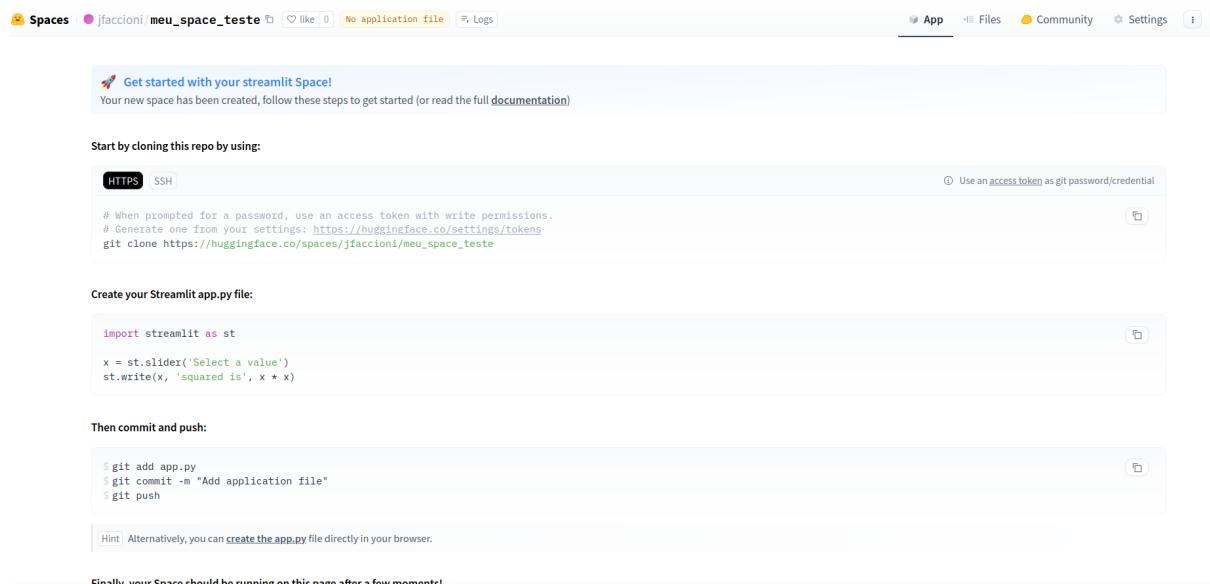
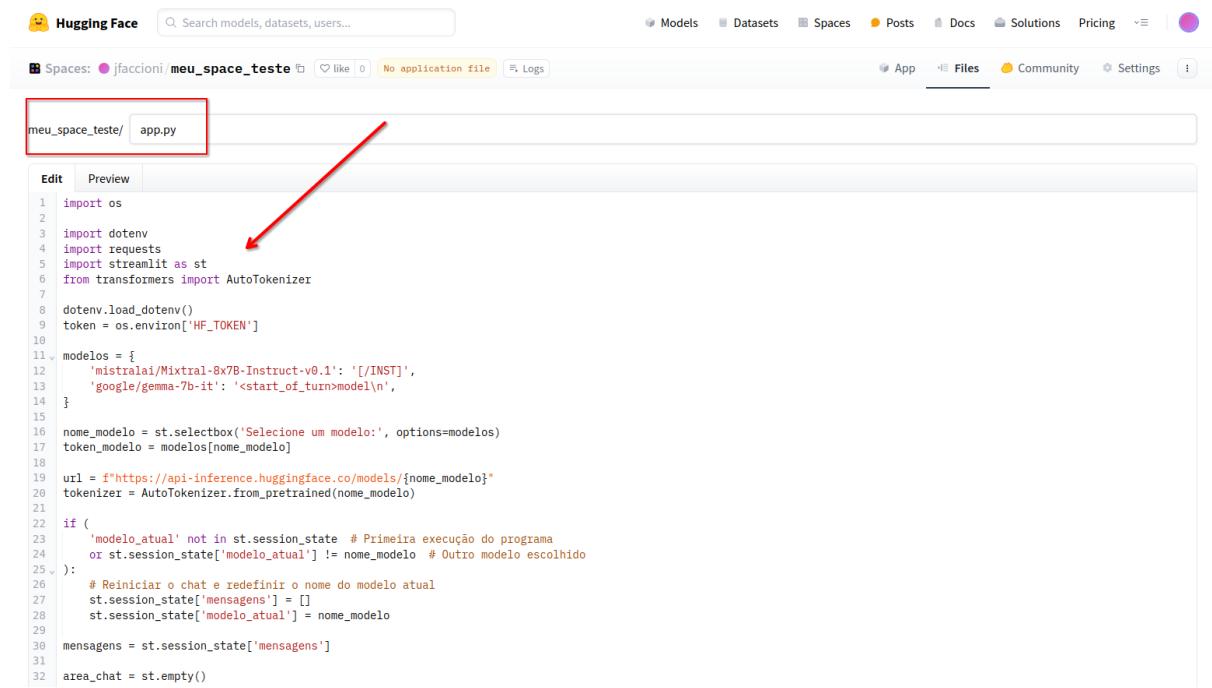


Figure 30: Space recém criado

As instruções explicam como sincronizar o código do Space com algum repositório git onde você tenha o código. Mas neste curso, vamos usar a opção alternativa de escrever o arquivo app.py diretamente no browser. Clique no link onde está grifado `create the app.py`, na última frase da imagem acima.

Ao clicar no link, você verá um editor de texto. Basta copiar e colar todo o seu código para este editor, conforme imagem abaixo:

Explorando o Universo das IAs com Hugging Face



The screenshot shows the Hugging Face platform's interface. At the top, there's a navigation bar with links for Models, Datasets, Spaces, Posts, Docs, Solutions, Pricing, and a user profile. Below the navigation is a search bar and a breadcrumb trail showing the current space: 'Spaces: jfaccioni/meu_space_teste'. The main area is a code editor titled 'app.py' under the path 'meu_space teste/'. The code itself is a Python script for a Streamlit application. It imports os, dotenv, requests, and streamlit, and uses AutoTokenizer from transformers. It loads environment variables, defines a dictionary of models, and uses a selectbox to choose a model. It then constructs a URL for the API inference endpoint and initializes a Streamlit session state. The code ends with an empty area_chat variable.

```
1 import os
2
3 import dotenv
4 import requests
5 import streamlit as st
6 from transformers import AutoTokenizer
7
8 dotenv.load_dotenv()
9 token = os.environ['HF_TOKEN']
10
11 modelos = {
12     'mistralai/Mixtral-8v7B-Instruct-v0.1': '[/INST]',
13     'google/gemma-7b-it': '<start_of_turn>model\n',
14 }
15
16 nome_modelo = st.selectbox('Selecione um modelo:', options=modelos)
17 token_modelo = modelos[nome_modelo]
18
19 url = f"https://api-inference.huggingface.co/models/{nome_modelo}"
20 tokenizer = AutoTokenizer.from_pretrained(nome_modelo)
21
22 if (
23     'modelo_atual' not in st.session_state # Primeira execução do programa
24     or st.session_state['modelo_atual'] != nome_modelo # Outro modelo escolhido
25 ):
26     # Reiniciar o chat e redefinir o nome do modelo atual
27     st.session_state['mensagens'] = []
28     st.session_state['modelo_atual'] = nome_modelo
29
30 mensagens = st.session_state['mensagens']
31
32 area_chat = st.empty()
```

Figure 31: Editando o arquivo app.py.

E em seguida, clicar em Commit new file to main (assegure que a opção commit directly to main esteja selecionada):

Explorando o Universo das IAs com Hugging Face



Figure 32: Salvando o código no seu Space

Passo 3: Adicione as dependências

Seu código está no ar! Vamos checar seu aplicativo na aba App:

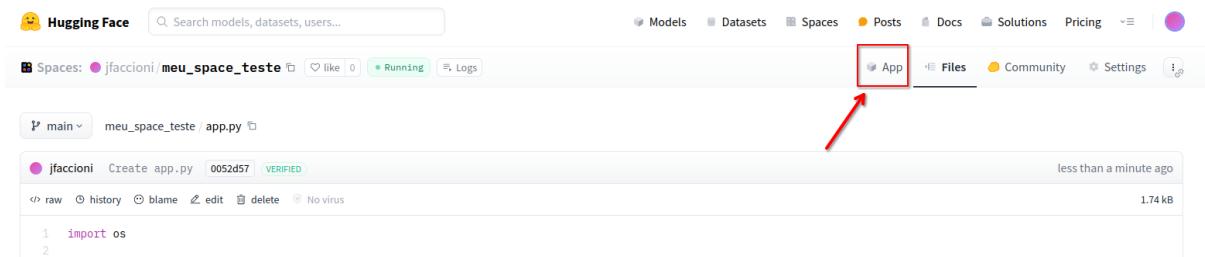


Figure 33: Selecionando a aba App do seu Space.

Contudo, seu aplicativo deve estar com um erro (você pode clicar em Logs no topo da tela para ter a informação completa do terminal):

Explorando o Universo das IAs com Hugging Face

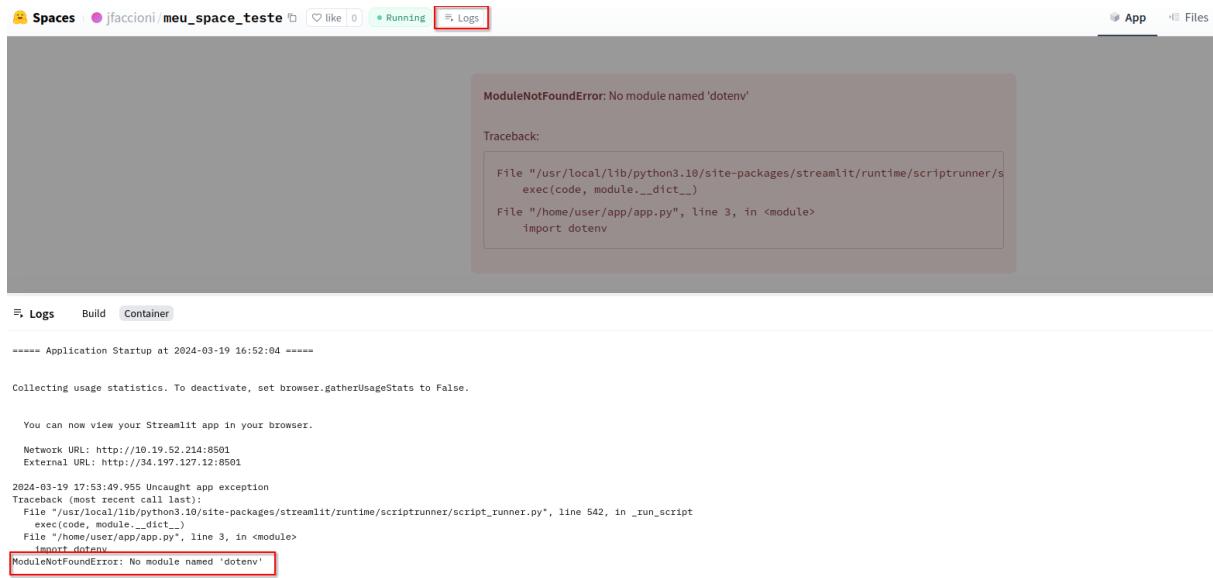


Figure 34: Erro no aplicativo

O problema aqui é que nunca especificamos as dependências (bibliotecas) que seu webapp precisa. Para fazer isso, volte para a aba **Files**, e clique em **Add file** e em seguida **Create a new file** (note também que alguns arquivos de configuração foram criados pelo Hugging Face, além do seu `app.py`):

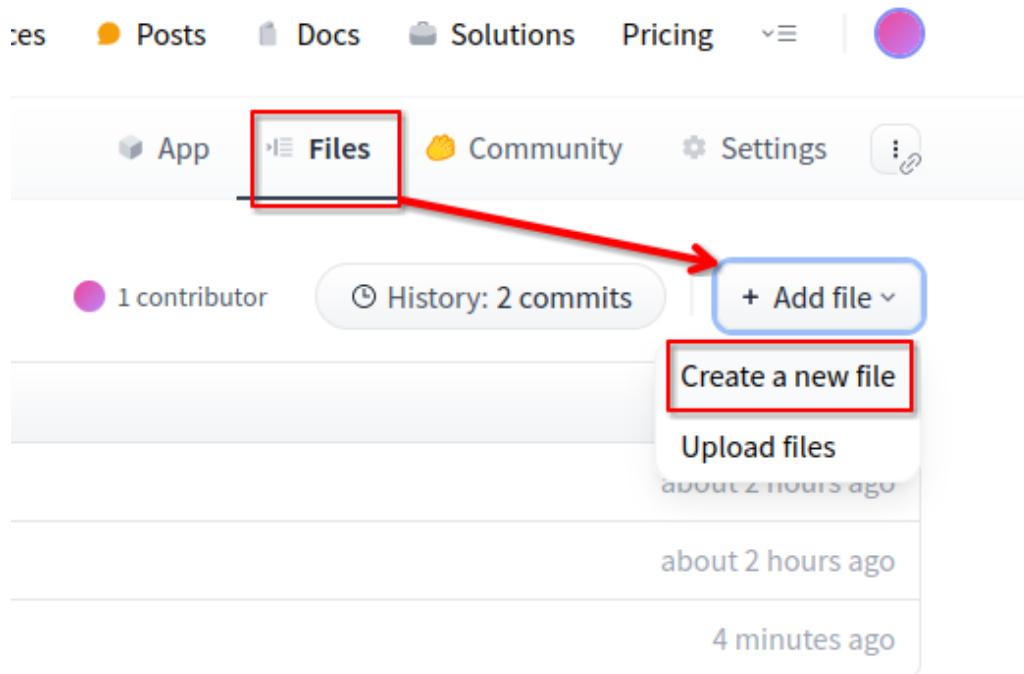


Figure 35: Criando um novo arquivo

Chame o arquivo de `requirements.txt` (outros nomes não irão funcionar) e declare as dependências abaixo, uma por linha:

```
streamlit
python-dotenv
requests
transformers
```

Note que poderíamos ter especificado a versão das bibliotecas com o formato `streamlit==1.32.1`, mas como não fizemos isso as versões mais recentes serão usadas.

Salve novamente o arquivo clicando em `Commit new file to main`:

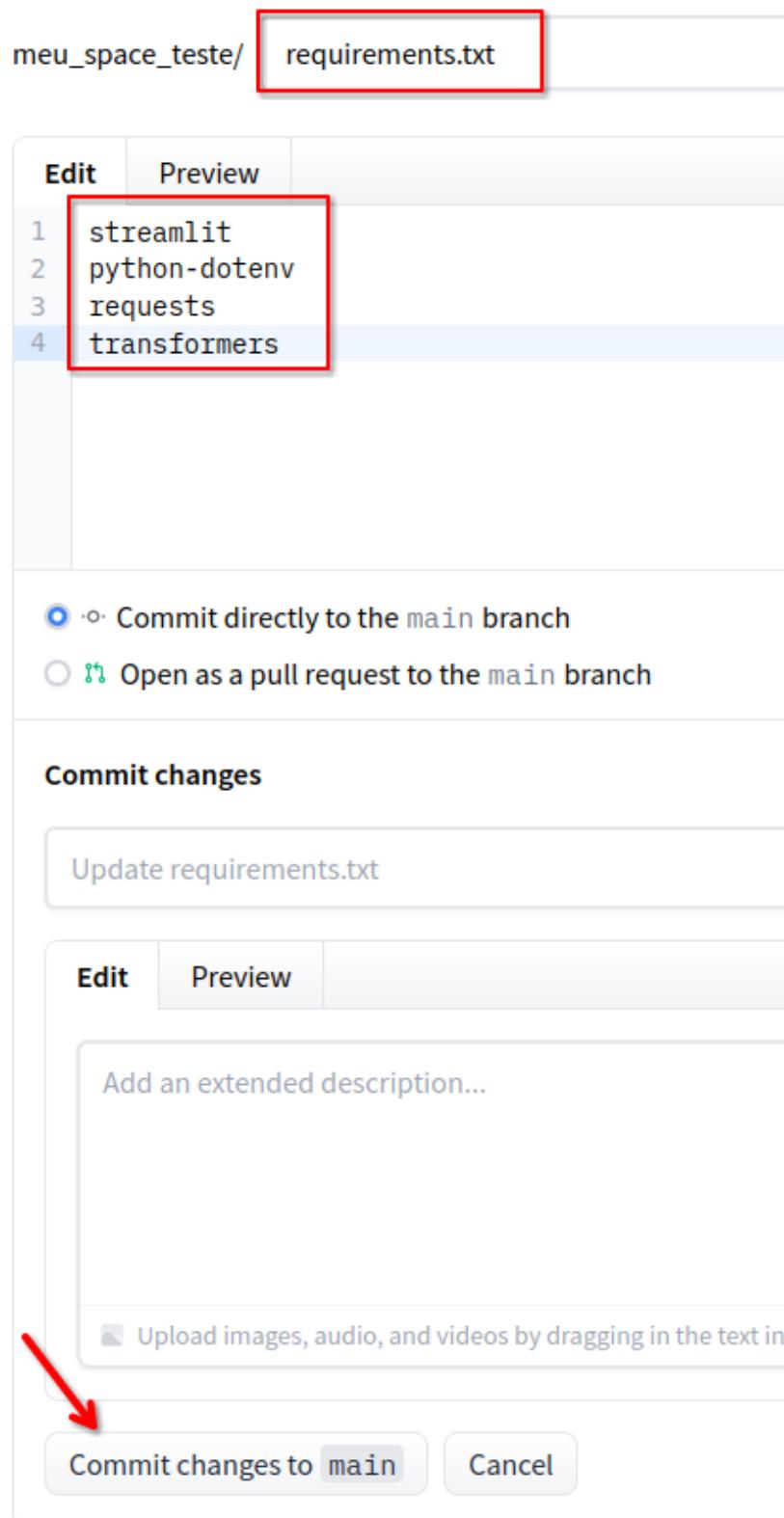


Figure 36: Criando o arquivo requirements.txt

Esse processo irá automaticamente disparar um novo processo de deploy do seu webapp. Você verá o status dele como Building ao invés de Running.

Passo 4: gerindo o token do Hugging Face

Quase lá! O seu app deve gerar um novo erro agora:

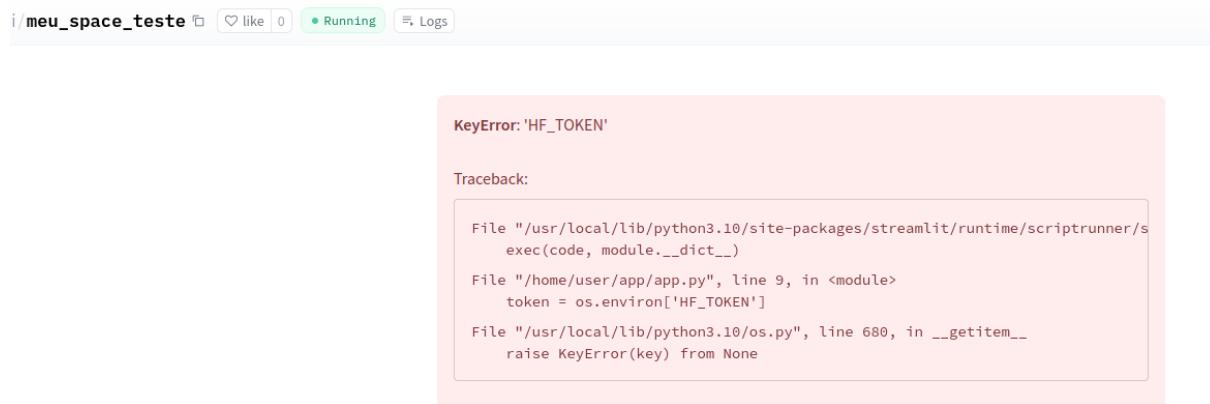


Figure 37: Novo erro gerado pelo app

O problema é que nunca adicionamos nosso token ao aplicativo (lembrando que precisamos desse token para rodar os modelos restritos do nosso webapp).

Contudo, **não queremos sob hipótese alguma subir nosso arquivo .env para o repositório**, sobre tudo se o repositório estiver aberto!

Para adicionar um token de forma segura, vamos até o painel de configuração do webapp (aba Settings):

Explorando o Universo das IAs com Hugging Face

The screenshot shows the 'Space Hardware' section of the Hugging Face Settings page. It displays various hardware options with their prices per hour:

| Hardware | VCPU | RAM | VRAM | Price |
|---------------------|---------|------------|------------|--------------|
| CPU basic | 2 vCPU | 16 GB RAM | - | \$0.03/hour |
| CPU upgrade | 8 vCPU | 32 GB RAM | - | \$0.03/hour |
| Nvidia T4 small | 4 vCPU | 15 GB RAM | 16 GB VRAM | \$0.60/hour |
| Nvidia T4 medium | 8 vCPU | 30 GB RAM | 16 GB VRAM | \$0.90/hour |
| Nvidia A10G small | 4 vCPU | 15 GB RAM | 24 GB VRAM | \$1.05/hour |
| Nvidia A10G large | 12 vCPU | 46 GB RAM | 24 GB VRAM | \$3.15/hour |
| Nvidia A100 large | 12 vCPU | 142 GB RAM | 40 GB VRAM | \$4.13/hour |
| Nvidia 2xA10G large | 24 vCPU | 92 GB RAM | 48 GB VRAM | \$5.70/hour |
| Nvidia 4xA10G large | 48 vCPU | 184 GB RAM | 96 GB VRAM | \$10.80/hour |
| Nvidia H100 | 24 vCPU | 250 GB RAM | 80 GB VRAM | \$8.70/hour |

Below the hardware options, there are sections for 'Sleep time settings' (set to 48 hours) and 'Pause Space' (disabled). A note at the bottom encourages applying for a community GPU grant.

Figure 38: Área de Settings do Space.

Em seguida, vá até o tópico `Variables and secrets`, clique em `New secret` e adicione o valor do seu token (dentro do seu arquivo local `.env`). Lembre-se de dar o mesmo nome que é usado pelo código do webapp:

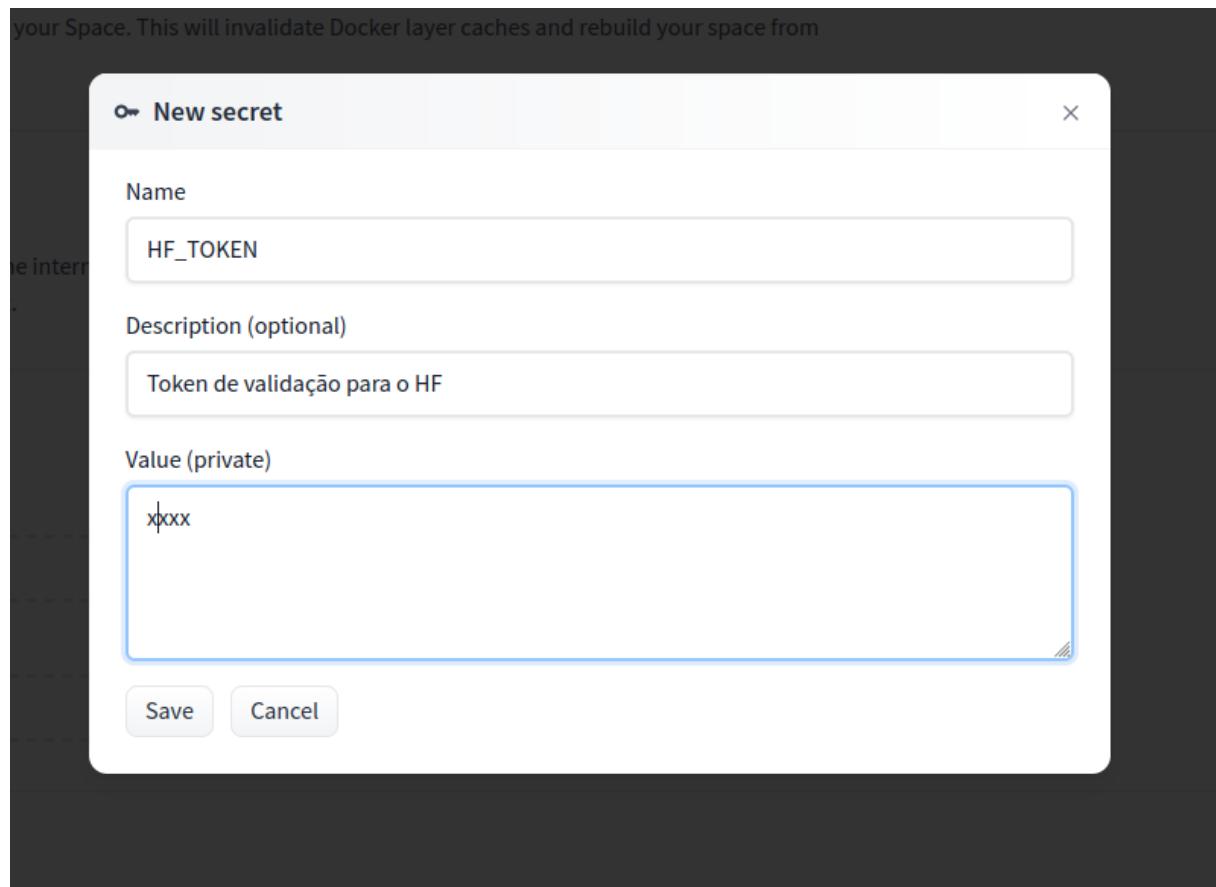


Figure 39: Adicionando o token

Ao salvar o token, um novo processo de deploy será disparado. Ao final, devemos ter nosso webapp funcional. **Sucesso!**

Explorando o Universo das IAs com Hugging Face

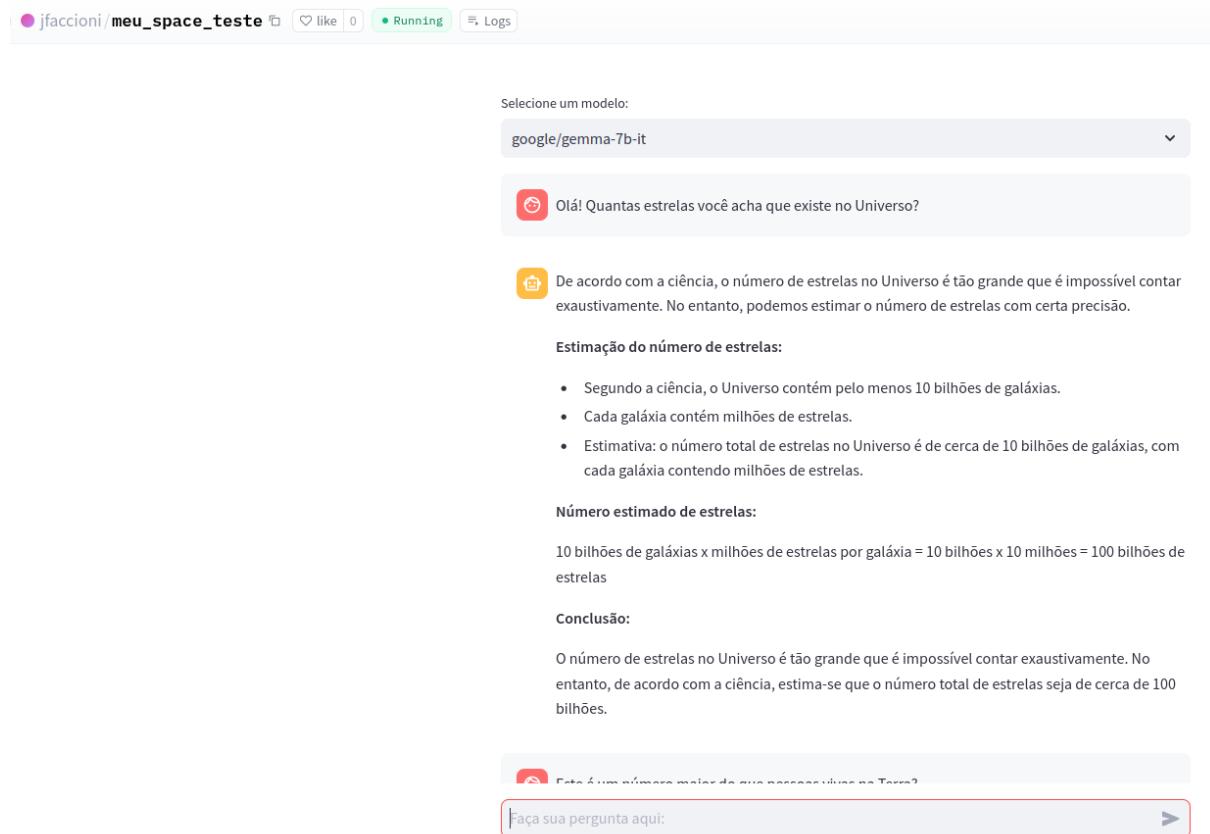


Figure 40: Interface do webapp funcional no nosso Space do Hugging Face.

18. Considerações sobre o deploy e encerramento

Configurações do Deploy

Há alguns pontos a se considerar sobre o nosso Deploy.

Hardware utilizado

Na aba de configuração, é possível avaliar outros hardwares possíveis para rodar seu webapp (mas apenas o nível mais básico é gratuito). Também é possível configurar acesso a disco rígido persistente, caso você queira manter dados salvos.

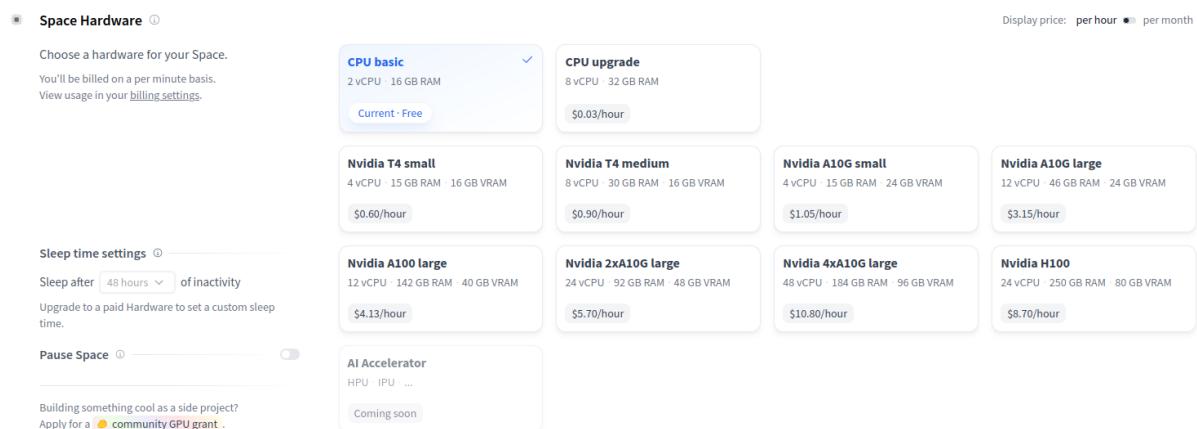


Figure 41: Outras opções de configuração no Space.

Lembre-se que, se os modelos estiverem rodando localmente no seu webapp (isto é, sem usar a Inference API), pode ser que o hardware padrão não seja suficiente. É importante testar antes!

Visibilidade do Space

Seu Space está publicado no link https://huggingface.co/spaces/seu_usuario/seu_space/.

Publicamos nosso Space no modo Public - isso significa que **qualquer pessoa tem acesso tanto ao webapp, quanto ao seu código fonte** (você pode testar isso acessando o link no modo anônimo do seu navegador).

Você pode torná-lo privado a qualquer momento nas configurações, porém isso fará com que apenas você tenha acesso (tanto ao webapp quanto aos arquivos).

Comunidade

Se o seu Space for público, outros membros da comunidade podem criar discussões (ou até mesmo enviar *pull requests*, que é uma forma de sugerir modificações no seu código). Essas informações ficam na aba Community. Essas contribuições podem ser configuradas nos Settings.

Você também consegue ver os seus Spaces publicados na página do seu perfil pessoal (além de editar outros detalhes).

Compartilhamento

No menu com “três pontinhos” à direita da aba Settings, há opções de compartilhamento, como a possibilidade de embarcar este Space em outro site através de um *iframe*, adicionar a uma coleção de Spaces, ou até mesmo conferir quais modelos são utilizados:

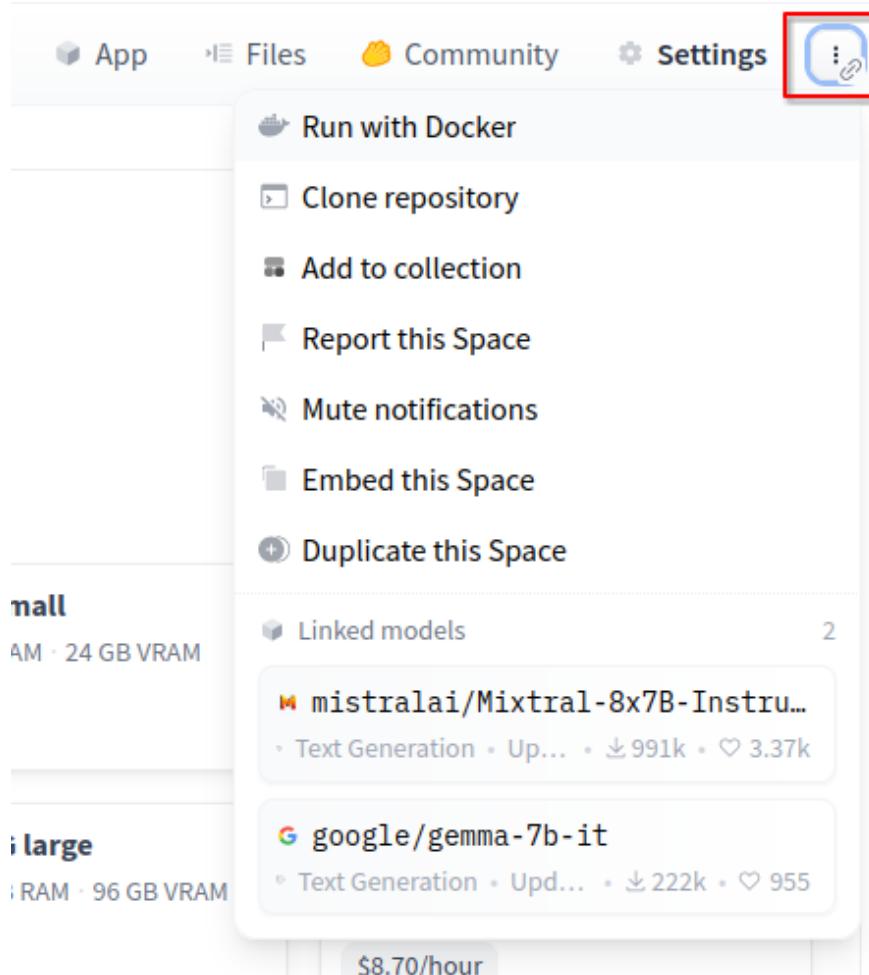


Figure 42: Opções adicionais para o Space.

Outras opções

Ao final da página de `Settings`, você pode acompanhar o uso/popularidade do seu Space (seção `Analytics`), ou então deletar o seu Space.

Logs

Por fim, não se esqueça de consultar os logs do seu Space! Se qualquer coisa der errado, este é o local onde haverá alguma mensagem de erro. Os logs de `build` são referentes ao deploy que é iniciado a cada modificação. Já os logs de `container` equivalem às mensagens no terminal que aparecem quando você executa o código.

Encerramento

Neste curso, aprendemos como utilizar a plataforma Hugging Face. Aprendemos como fazer para:

- Baixar modelos de IA utilizando a biblioteca `transformers`.
- Acessar modelos de IA através da Inference API.
- Acessar conjuntos de dados depositados no Hugging Face pela biblioteca `datasets`.
- Criar nosso próprio webapp utilizando modelos de IA e fazer seu deploy em um Space do Hugging Face.

É claro que o Hugging Face não acaba aqui. Não tivemos tempo de explorar boa parte das tarefas que existem no Hugging Face. Dito isso, agora sabemos *como* fazer este processo: conseguimos procurar por uma tarefa nova, encontrar o código necessário para baixá-lo (seja pela biblioteca `transformers` ou alguma outra), avaliar se conseguiremos rodar o modelo localmente, e entender como passar argumentos e obter resultados a partir dele.

Com isso, você está apto para combinar funcionalidades de diversos modelos do Hugging Face com suas aplicações. Não é preciso utilizar o Hugging Face para fazer tudo em uma aplicação: você pode apenas pegar um ou alguns modelos específicos que sirvam para alguma tarefa, e combiná-los com algum outro script ou programa. Ou então combinar as IAs do Hugging Face com outras IAs mais “famosas”, como o ChatGPT, e construir assim um sistema robusto com diversas IAs para uma variedade de tarefas. O céu é o limite!