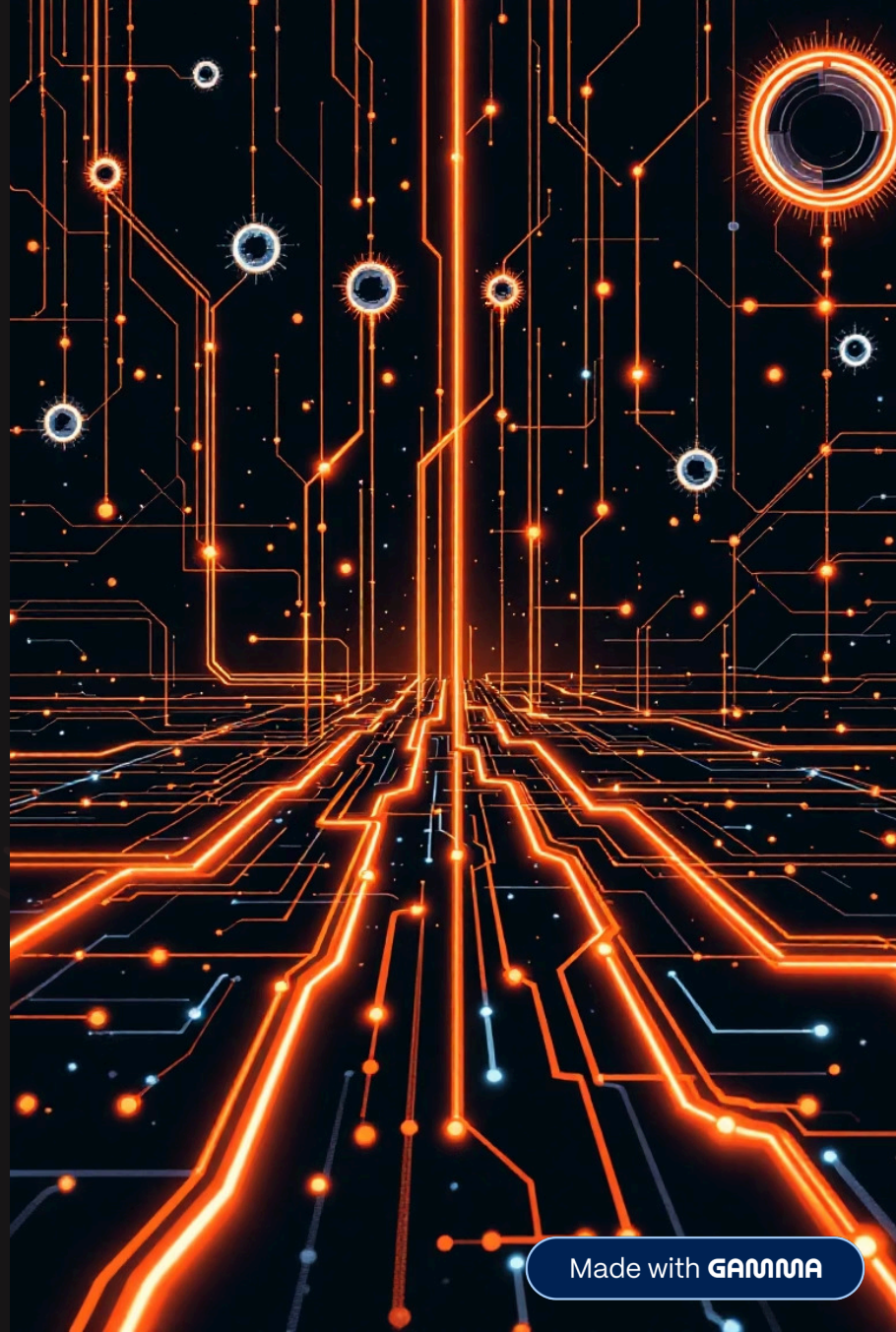
# Comparação de Modelos de Linguagem com Hugging Face

Experimento prático comparando dois LLMs distintos utilizando Hugging Face no Google Colab



# Modelos Analisados e Processo

## distilbert/distilgpt2

- Modelo leve e rápido
- Compatível com CPU
- Executou normalmente
- Respostas coerentes geradas

## openai/gpt-oss-20b

- Modelo muito maior
- Exige GPU com BF16
- Carregou mas não gerou texto
- Incompatibilidade de dtype

01

Instalação de bibliotecas

02

Carregamento dos modelos

03

Geração e comparação

04

Análise de resultados

Made with GAMMA

# Resultados e Conclusões

**107**

**Caracteres gerados**

distilgpt2 produziu resposta curta e coerente

**6**

**Palavras na saída**

Respostas simples mas efetivas

**0**

**Saída do gpt-oss-20b**

Falha por limitação de hardware

### Objetivo cumprido

Aplicação comparou modelos e expôs limitações práticas reais

### Hardware é crucial

Modelos grandes exigem GPU A100 para execução adequada

### Expansão possível

Métricas automáticas e testes com mais modelos

Made with GAMMA

# Anexos

```
[5]
✓ 10s
!pip install -q transformers accelerate torch sentencepiece
```

```
[6]
✓ 0s
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

device = "cuda" if torch.cuda.is_available() else "cpu"
print("Usando dispositivo:", device)
```

```
Usando dispositivo: cpu
```

```
[4]
✓ 0s
model_names = {
    "distilgpt2": "distilbert/distilgpt2",
    "gpt_oss_20b": "openai/gpt-oss-20b"
}
```

```
[7]
tokenizers = {}
models = {}

for apelido, nome in model_names.items():
    print(f"\n=== Carregando {apelido} ({nome}) ===")
    # Tokenizer
    tokenizer = AutoTokenizer.from_pretrained(nome)

    # Alguns modelos não têm pad_token definido
    if tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token

    tokenizers[apelido] = tokenizer

    # Modelo
    if apelido == "gpt_oss_20b":
        # Modelo grande: usar device_map e dtype reduzido
        model = AutoModelForCausalLM.from_pretrained(
            nome,
            torch_dtype=torch.bfloat16 if torch.cuda.is_available() else torch.float32,
            device_map="auto"
        )
    else:
        # Modelo menor: pode ir direto pra GPU se tiver
        model = AutoModelForCausalLM.from_pretrained(nome)
        model = model.to(device)

    models[apelido] = model

print("\nModelos carregados com sucesso (se não deu erro acima 😅).")
```

```
=== Carregando distilgpt2 (distilbert/distilgpt2) ===
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 423B/s]
config.json: 100% 762/762 [00:00<00:00, 12.2kB/s]
vocab.json: 1.04M/? [00:00<00:00, 2.92MB/s]
merges.txt: 456k/? [00:00<00:00, 3.88MB/s]
tokenizer.json: 1.36M/? [00:00<00:00, 7.51MB/s]
model.safetensors: 100% 353M/353M [00:05<00:00, 51.2MB/s]
generation_config.json: 100% 124/124 [00:00<00:00, 1.51kB/s]
```

```
=== Carregando gpt_oss_20b (openai/gpt-oss-20b) ===
tokenizer_config.json: 4.20k/? [00:00<00:00, 50.5kB/s]
tokenizer.json: 100% 27.9M/27.9M [00:00<00:00, 50.5MB/s]
special_tokens_map.json: 100% 98.0/98.0 [00:00<00:00, 2.18kB/s]
chat_template.jinja: 16.7k/? [00:00<00:00, 416kB/s]
config.json: 1.81k/? [00:00<00:00, 38.3kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
Using MXFP4 quantized models requires a GPU, we will default to dequantizing the model to bf16
model.safetensors.index.json: 36.4k/? [00:00<00:00, 905kB/s]
Fetching 3 files: 0% 0/3 [00:00<?, ?it/s]
model-00000-of-00002.safetensors: 94% 4.53G/4.79G [02:36<00:06, 41.1MB/s]
model-00001-of-00002.safetensors: 57% 2.74G/4.80G [02:37<00:42, 48.8MB/s]
model-00002-of-00002.safetensors: 100% 4.17G/4.17G [01:55<00:00, 8.72MB/s]
```

```
def gerar_texto(model_key, prompt, max_new_tokens=80, temperature=0.8, top_p=0.9):
    tokenizer = tokenizers[model_key]
    model = models[model_key]

-   inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
+   # Ensure input tensor dtype matches model dtype
+   inputs = tokenizer(prompt, return_tensors="pt").to(model.device).to(model.dtype)

    with torch.no_grad():
        output = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            do_sample=True,
            top_p=top_p,
            temperature=temperature,
            pad_token_id=tokenizer.eos_token_id
        )
-
+
    texto = tokenizer.decode(output[0], skip_special_tokens=True)
    return texto


def comparar_modelos(prompt, max_new_tokens=80):
    resultados = {}
    for apelido in model_names.keys():
        try:
            print(f"\n>>> Gerando com modelo: {apelido}")
            resposta = gerar_texto(
                model_key=apelido,
                prompt=prompt,
                max_new_tokens=max_new_tokens
            )
            resultados[apelido] = resposta
        except Exception as e:
            resultados[apelido] = f"Erro ao gerar com {apelido}: {e}"
-
+
```

```
def analisar_resultados(prompt, resultados):
    print("\nAnálise simples das respostas:")
    for modelo, texto in resultados.items():
        num_caracteres = len(texto)
        num_palavras = len(texto.split())
        print(f"- {modelo}: {num_caracteres} caracteres, {num_palavras} palavras.")

# Definir a lista de prompts antes de usá-la
prompts = [
    "Qual é a capital da França?",
    "Me conte uma história curta sobre um robô que se apaixona."
]

# Exemplo usando apenas o primeiro prompt:
prompt_teste = prompts[0]
print("Prompt de teste:", prompt_teste)

resultados_teste = comparar_modelos(prompt_teste, max_new_tokens=80)
for modelo, resposta in resultados_teste.items():
    print(f"\n[{modelo}]")
    print(resposta[:400], "\n")  # corta pra não ficar enorme

analisar_resultados(prompt_teste, resultados_teste)
```

```
Attempting to cast a BatchEncoding to type torch.float32. This is not supported.
Prompt de teste: Qual é a capital da França?

>>> Gerando com modelo: distilgpt2
Attempting to cast a BatchEncoding to type torch.float32. This is not supported.

>>> Gerando com modelo: gpt_oss_20b

[distilgpt2]
Qual é a capital da França?
```

```
[gpt_oss_20b]
Erro ao gerar com gpt_oss_20b: expected m1 and m2 to have the same dtype, but got: float != c10::BFloat16


Análise simples das respostas:
- distilgpt2: 107 caracteres, 6 palavras.
- gpt_oss_20b: 105 caracteres, 19 palavras.
```

Made with GAMMA