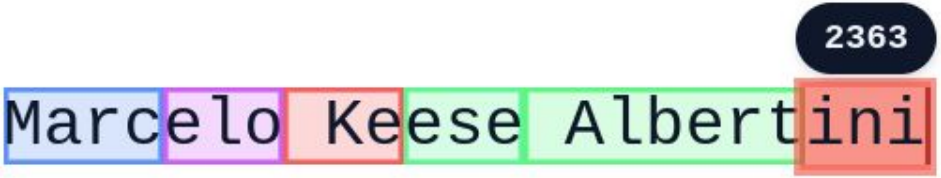
# Aula 1: Tokenização




Prof. Marcelo Keese Albertini

Universidade Federal de Uberlândia

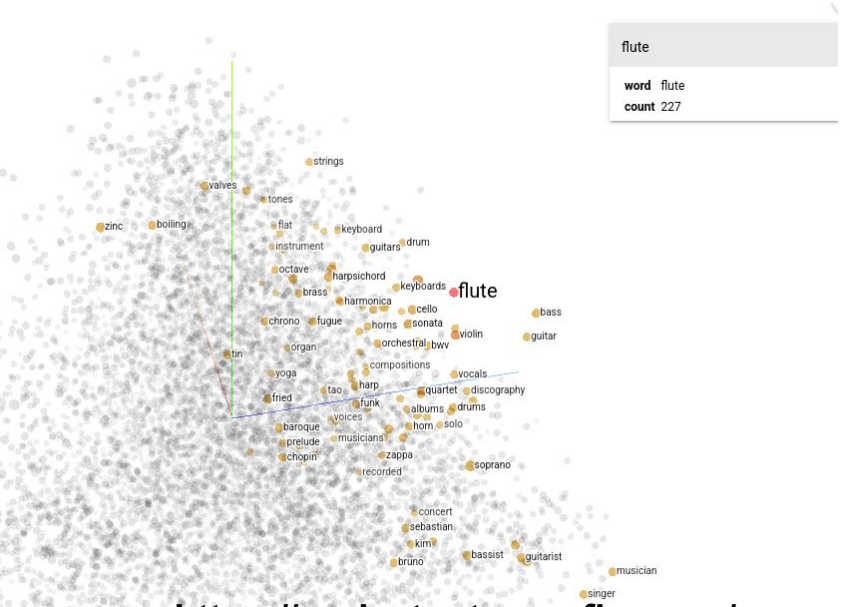
# Roteiro

1. O que é tokenização
2. Papel da tokenização em LLM
3. Pipeline de tokenização

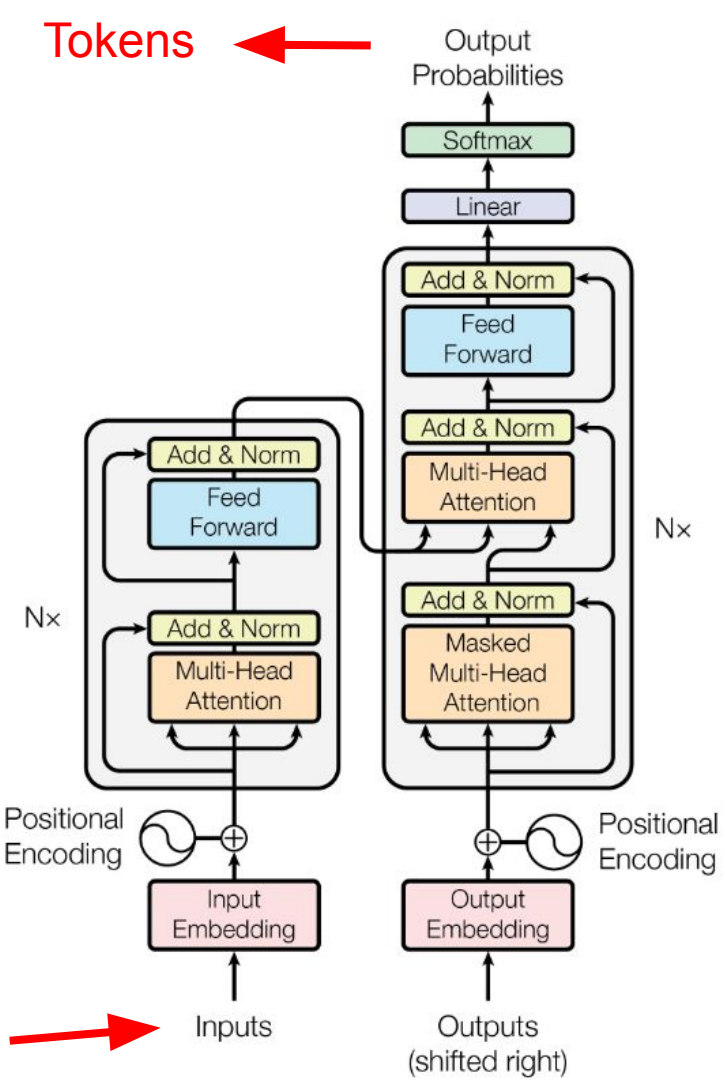
# Transformer

"Attention Is All You Need" (2017)

https://arxiv.org/abs/1706.03762




https://projector.tensorflow.org/



# Tokenização

Elo entre texto e modelo: transforma linguagem natural em unidades processáveis.

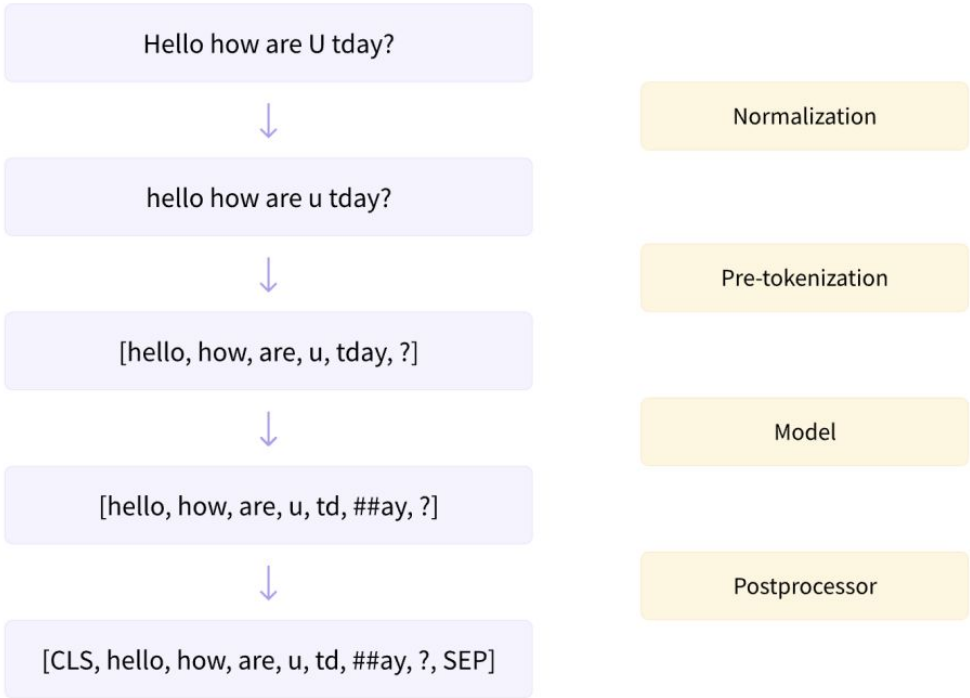
```
gpt-tokenizer is the fastest TypeScript implementation of
OpenAI's tokenizers.
It supports GPT-5, GPT-4.1, o-series models, streaming
encoding, chat prompts, cost estimation and much more.
```

https://gpt-tokenizer.dev/

```
Frase: "Gatos são legais!"
Tokens: ["Gatos", "são", "legais", "!"]
IDs: [1532, 211, 897, 33]
```

# Pipeline de tokenização

- Normalização
- Pré-tokenização
- Modelagem
- Pós-processamento




https://lecture.jeju.ai/lectures/nlp_deep/tokenization/pipeline.html

# Normalização

- Conversão de texto em formato padrão

- Normalização Unicode

```
normalizer = normalizers.Sequence([
    NFD(),           # decomposição de acentos
    Lowercase(),     # tudo minúsculo
    StripAccents()   # remove acentos
])
texto = "Héllò hôw are ü?"
print("Antes:", texto)
print("Depois:", normalizer.normalize_str(texto), "\n")
tokenizer.normalizer = normalizer
```

```
Antes: Héllò hôw are ü?
Depois: hello how are u?
```

https://unreliable.io/unicode-normalization/

# Pré-tokenização

Define unidades básicas de tokenização, ou seja, **como o texto será quebrado** — por espaço, pontuação, ou outras regras.

- Objetivo é **garantir que os tokens não cruzem fronteiras de palavras**.

```
print("# Pre-tokenization")
pre_tok = Sequence([
    Whitespace(),
    Digits(individual_digits=True)
])
texto2 = "Hello! How are you? Tenho R$ 213,12."
print("Pré-tokenização:", pre_tok.pre_tokenize_str(texto2), "\n")
tokenizer.pre_tokenizer = pre_tok
```

```
# Pre-tokenization
Pré-tokenização: [('Hello', (0, 5)), ('!', (5, 6)), ('How', (7, 10)), ('are', (11, 14)), ('you',
```

```python
from tokenizers import Tokenizer, models, pre_tokenizers

tok_ws = Tokenizer(models.BPE())
tok_ws.pre_tokenizer = pre_tokenizers.Whitespace()
frase = "Não, será punido o criminoso."

print(tok_ws.pre_tokenizer.pre_tokenize_str(frase))
```

```
[('Não', (0, 3)), (',', (3, 4)), ('será', (5, 9)), ('punido', (10, 16)), ('o', (17, 18)), ('criminoso', (19, 28)), ('.', (28, 29))]
```

## Metaspace (SentencePiece style)

```python
tok_meta = Tokenizer(models.BPE())
tok_meta.pre_tokenizer = pre_tokenizers.Metaspace()
print(tok_meta.pre_tokenizer.pre_tokenize_str(frase))
```

```
[('_Não,', (0, 4)), ('_será', (4, 9)), ('_punido', (9, 16)), ('_o', (16, 18)), ('_criminoso.', (18, 29))]
```

# Pós-processamento

- O modelo converte **tokens** em **IDs** aprendidos pelo **BPE**.
- O pós-processador adiciona **tokens especiais** conforme a tarefa:

a. `[CLS]` → início da sequência
b. `[SEP]` → separador entre frases (usado em pares)
c. Máscara de atenção etc.

```
# ------------------------------------------------------------
# Post-processing
# ------------------------------------------------------------
print("# Post-processing (TemplateProcessing)")
tokenizer.post_processor = TemplateProcessing(
    single="[CLS] $A [SEP]",
    pair="[CLS] $A [SEP] $B:1 [SEP]:1",
    special_tokens=[("[CLS]", 1), ("[SEP]", 2)],
)
```

# Modelagem (Tokenização)

- Treino: construção do vocabulário
- Encoding: texto para tokens
- Decoding: tokens para texto

```
# ------------------------------------------------------------
# Aplicando tudo
# ------------------------------------------------------------
encoded = tokenizer.encode("olá mundo")
print("Tokens IDs:", encoded.ids)
print("Tokens:", encoded.tokens)
```

```
# Model: BPE (Byte Pair Encoding)
# Post-processing (TemplateProcessing)
Tokens IDs: [1, 16, 15, 1, 1, 1, 1, 11, 16, 2]
Tokens: ['[CLS]', 'o', 'l', '<unk>', '<unk>', '<unk>', '<unk>', 'd', 'o', '[SEP]']
```

# Tokenizadores modernos

**Tokenizadores modernos**:

- **ByteLevel BPE** (GPT-2: 50,257 tokens, GPT-3, GPT-4)
- **WordPiece (BERT)**
- **Unigram**
- **SentencePiece**

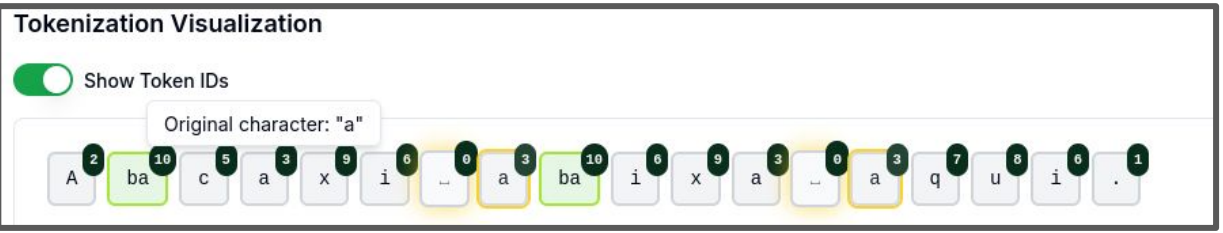
“Evolução: BPE (2016) → WordPiece (2018) → Unigram/SentencePiece (2018+)”

https://arxiv.org/html/2406.11214v2

# Tokenizador *Byte Pair Encoding (BPE)*

1. Inicia com 1 token por character
2. A cada iteração, combina os **pares de tokens mais frequentes**.

“Abacaxi abaixa aqui.”




https://www.bpe-visualizer.com

**Algorithm 1** Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

espaço

frequência da palavra

```
import re

bigram = "lo w"
p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')

text = "lo w est lo w er"
print(re.findall(p, text))
```

```
['lo w', 'lo w']
```

https://arxiv.org/pdf/1508.07909

https://lecture.jeju.ai/lectures/nlp_deep/tokenization/wordpiece.html

# Tokenizador *WordPiece*

- Algoritmo de treino
  a. Pré-tokenização: espaços em branco
  b. Vocabulário inicial: letras iniciais de palavras e as letras em palavras com prefixos ##
  c. Adiciona caracteres especiais
     - ["[PAD]", "[UNK]", "[CLS]", "[SEP]", "[MASK]"]
  d. Une sucessivamente pares com pontuação alta
     - Pontuação:

$$\text{score}(x, y) = \frac{p(xy)}{p(x)p(y)}$$

     - Se palavra 2 começar com #, ignorar # e adicionar como prefixo depois

```
['investment', 'opportunities', 'in', 'the', 'c', '##o', '##m', '##p', '##a', '##n', '##y']
```

https://lecture.jeju.ai/lectures/nlp_deep/tokenization/wordpiece.html

# Tokenizador *WordPiece*

- Encoding - algoritmo **greedy**
  a. Busca sucessivamente maior prefixo no vocabulário
  b. Se prefixo for interno, inicia com “##”

```
def encode_word(word):
    tokens = []
    while len(word) > 0:
        i = len(word)
        while i > 0 and word[:i] not in vocab:
            i -= 1
        if i == 0:
            return ["[UNK]"]
        tokens.append(word[:i])
        word = word[i:]
        if len(word) > 0:
            word = f"##{word}"
    return tokens
```

- Algoritmo ótimo: programação dinâmica (algoritmo de Viterbi)
  - busca pela sequência mais provável (P(t) = frequência de t / total )

$$\text{melhor tokenização} = \arg \min_{\text{segmentação}} \sum_i (-\log P(t_i))$$

https://lecture.jeju.ai/lectures/nlp_deep/tokenization/wordpiece.html

# Tokenizador *WordPiece*

- Algoritmo: decoding

```
def decode_wordpiece(tokens):
    palavras = []
    atual = ""
    for t in tokens:
        if t in ["[CLS]", "[SEP]", "[PAD]", "[MASK]"]:
            continue
        if t.startswith("##"):
            atual += t[2:]
        else:
            if atual:
                palavras.append(atual)
            atual = t
    if atual:
        palavras.append(atual)
    return " ".join(palavras)
```

# Avaliação de tokenizadores

- Critérios:
  - Tokenização ruim → sequências tokenizadas longas e lentas.
  - Vocabulário mal ajustado → muitos `<unk>`.
  - Normalização inconsistente → reconstrução incorreta.
  - Reversibilidade (decode igual a original)

# Métricas

- Tokens por char (compactação)
- Tokens por palavra (fragmentação)
- Cobertura: 1 - Taxa de <UNK> (robustez lexical)
- Comprimento médio de sequências (custo computacinal)
- Taxa de reversibildade

```
tpc = (df["tokens"] / df["chars"]).mean()
tpw = (df["tokens"] / df["words"]).mean()
unk_rate = (df["unk"].sum() / df["tokens"].sum()) * 100
decode_acc = (df["decoded_ok"].mean()) * 100
```

# Prática sobre tokenização

Use como base
https://colab.research.google.com/drive/1bGsDzvHXTmOJdv-n5T4c2Pt7nh_k01x7?usp=sharing

1. Definir um tema próprio para trabalhar nas próximas aulas
    a. Exemplos: músicas, tradução, resumo,...
2. Criar corpus ( |D| > 100 docs)
    a. Baixar datasets, documentos ou site sobre tema escolhido no repo
    b. Guardar no repositório em arquivo JSONL onde cada linha representa um documento
3. Aplicar treino de tokenizador BPE e WordPiece
4. Avaliar comparativamente os 2 tokenizadores

# Referências


- https://everdark.github.io/k9/notebooks/ml/natural_language_understanding/subword_units/subword_units.nb.html
- https://github.com/google/sentencepiece
- https://zackproser.com/demos/tokenize
- Subword tokenization (clássico) — BPE para NMT (Sennrich et al., 2016): https://arxiv.org/abs/1508.07909
- SentencePiece (unigram LM) — paper + implementação: https://arxiv.org/abs/1808.06226
- Tokenizers (Hugging Face) — docs oficiais (treino, pré/pós-processamento): https://huggingface.co/docs/tokenizers/index
- Leaderboard de embeddings — MTEB (Hugging Face Space): https://huggingface.co/spaces/mteb/leaderboard
- Discussão — “Byte Pair Encoding is Suboptimal for LM Pretraining” (2020): https://arxiv.org/abs/2004.03720
- Unigram - https://huggingface.co/docs/course/en/chapter6/7
    - https://colab.research.google.com/github/huggingface/notebooks/blob/master/course/en/chapter6/section7.ipynb