Capítulo 13 – A Mecânica Transformer

Aléssio Miranda Jr.

Setembro - 2026/2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

O que você verá neste capítulo

A arquitetura Transformer, introduzida no artigo seminal *"Attention is All You Need"* de Vaswani et al. (2017), é o alicerce de todos os modelos de linguagem modernos — do BERT ao GPT-4, do LLaMA ao Gemini. Neste capítulo, desmontaremos a mecânica interna do Transformer peça por peça: a **Tokenização** (como o texto bruto é convertido em IDs numéricos), o **Positional Encoding** (como o modelo preserva a noção de ordem), e o **Self-Attention** (o mecanismo que permite ao modelo capturar dependências de longa distância entre qualquer par de palavras em uma frase).

1 Introdução: O Paradigma Antes de 2017

Até o final de 2017, a inteligência artificial para processamento de linguagem era dominada pelas Redes Neurais Recorrentes (RNNs) e suas variantes mais sofisticadas, as LSTMs (*Long Short-Term Memory*, Hochreiter & Schmidhuber, 1997). Esses modelos liam os textos da mesma forma que um ser humano: sequencialmente, da esquerda para a direita, uma palavra de cada vez.

O grande problema dessa abordagem era a **degradação de memória**. Se o modelo lesse um texto de 300 palavras, ao chegar na última palavra, ele já não "lembrava" direito quem era o sujeito principal citado na primeira linha. Além disso, a leitura sequencial **não pode ser paralelizada** na placa de vídeo (GPU), tornando o treinamento intoleravelmente lento para textos longos.

2 Tokenização: Convertendo Texto em Números

Antes de qualquer processamento, o texto bruto precisa ser convertido em sequências de números inteiros. Esse processo é a **Tokenização**. Mas diferentemente do Word2Vec (que tokenizava por palavras inteiras), os Transformers modernos usam algoritmos de **subpalavras** (*subword tokenization*):

- **BPE (Byte-Pair Encoding):** Usado pelo GPT. Começa com caracteres individuais e iterativamente funde os pares mais frequentes em tokens maiores. Palavras comuns como "the" viram um token único; palavras raras como "anticonstitucionalissimamente" são divididas em sub-tokens.
- **WordPiece:** Usado pelo BERT. Similar ao BPE, mas usa uma métrica de verossimilhança para decidir as fusões.
- **SentencePiece:** Usado pelo LLaMA e T5. Opera diretamente sobre bytes, sem necessidade de pré-tokenização por espaços. Funciona em qualquer idioma.

▷ Exemplo: title=Tokenização na Prática

O texto "Inteligência Artificial é fascinante" pode ser tokenizado pelo GPT como:

`['Intel', 'ig', 'ência', ' Art', 'ificial', ' é', ' fasc', 'inante']`

Cada sub-token recebe um ID numérico do vocabulário (ex: $[15496, 328, 25649, \ldots]$). Esse vetor de IDs é a entrada real do Transformer.

• Teoria: title=Por que Subpalavras e não Palavras Inteiras?

1. **Vocabulário controlado:** Em vez de um dicionário com milhões de palavras (incluindo conjugações, plurais, neologismos), o BPE mantém um vocabulário de ∼50.000 tokens que cobre qualquer texto.
2. **Palavras novas:** "ChatGPT", "COVID-19" e gírias são decompostos em sub-tokens conhecidos. Nenhuma palavra é "desconhecida".
3. **Eficiência:** Palavras frequentes consomem poucos tokens; palavras raras consomem mais. É uma codificação de comprimento variável ótima, análoga à compressão Huffman.

3 A Revolução: Attention is All You Need

Em junho de 2017, Vaswani, Shazeer, Parmar et al. publicaram o *paper* mais influente da história recente da IA: *"Attention is All You Need"*. A proposta

era radical: abandonar completamente as RNNs e LSTMs, e construir uma arquitetura baseada **exclusivamente** em mecanismos de atenção.

O que mudou? O Transformer ingere **todas as palavras** (tokens) de uma frase **ao mesmo tempo**, em paralelo. Isso permite treinar em GPUs massivamente paralelas e processar textos de milhares de tokens sem degradação de memória.

Mas se o modelo vê todas as palavras simultaneamente, como ele sabe a **ordem**? A resposta está em uma engenharia matemática elegante.

4 Positional Encoding: Injetando a Noção de Ordem

Como o Transformer não processa tokens sequencialmente, ele não tem noção intrínseca de que "gato" aparece antes de "dormiu" na frase. Para resolver isso, somamos ao Embedding de cada token um vetor de **Positional Encoding** — um sinal matemático que codifica a posição do token na sequência.

O artigo original utiliza funções senoidais de diferentes frequências:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{1}$$

Onde pos é a posição do token, i é a dimensão, e d é a dimensionalidade total do Embedding.

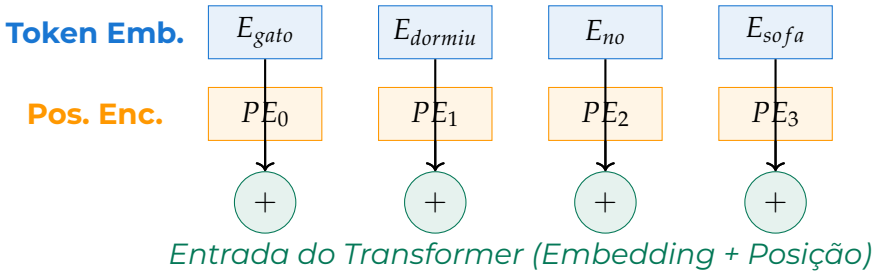


Figura 1: O Positional Encoding é somado ao Embedding de cada token antes da entrada no Transformer. Isso injeta a informação de posição sem alterar a arquitetura.

◇ Informação

A vantagem das funções senoidais é que o modelo pode generalizar para sequências mais longas do que viu durante o treinamento, pois os senos e cossenos geram padrões periódicos contínuos. Modelos modernos como GPT e LLaMA substituíram o encoding fixo por **RoPE** (*Rotary Position Embedding*), que é aprendido e mais flexível.

5 A Intuição do Self-Attention (Q, K, V)

O mecanismo de atenção é como uma sala cheia de pessoas (palavras) procurando seus pares para formar sentido. Ele usa três matrizes fundamentais baseadas no conceito de busca em banco de dados: **Query (Q)**, **Key (K)** e **Value (V)**.

- **Query (A Pergunta):** O que a palavra atual está procurando para dar sentido a si mesma?
- **Key (A Chave):** O que essa palavra tem a oferecer para as outras?
- **Value (O Valor):** A essência real da palavra (seu *Embedding* contextualizado).

Para cada token, o Transformer pega a **Query** dele e calcula o "encaixe" matemático com a **Key** de *todas as outras palavras da frase* (via Produto Escalar normalizado):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \tag{2}$$

O fator $\sqrt{d_k}$ é uma normalização que estabiliza os gradientes. Se a similaridade $Q \cdot K^T$ for alta entre dois tokens, o Transformer cria uma ponte invisível de **Atenção** entre eles.

▷ Exemplo: title=Resolução de Correferência via Atenção

Na frase *"O cachorro não cruzou a rua porque **ele** estava cansado"*, a rede calcula que o *Query* da palavra "ele" bate perfeitamente com a *Key* de "cachorro" (e não de "rua"). Assim, o Embedding contextualizado de "ele" absorve informação semântica de "cachorro", resolvendo automaticamente a correferência pronominal.

6 Múltiplas Cabeças (Multi-Head Attention)

Uma frase possui várias dimensões de sentido simultâneas: quem faz a ação, onde ocorre, o sentimento envolvido, a relação temporal. Para capturar tudo isso, o Transformer não roda a Atenção apenas uma vez. Ele roda h "Cabeças" (*heads*) independentes em paralelo.

Uma cabeça pode focar na **gramática** (concordância verbal), enquanto outra foca em **entidades** ("ele" $\rightarrow$ "cachorro"), e outra em **relações causais** ("porque" liga efeito a causa). No final, os resultados dessas visões transversais são concatenados e projetados linearmente.

Isso justifica por que os LLMs modernos dominam a linguagem: eles enxergam uma **teia invisível de dependências** entre todas as palavras de um

documento, permitindo um grau de compreensão de contexto insuperável por qualquer arquitetura anterior.

• Teoria: title=A Escala dos Transformers Modernos

- **BERT-base** (Google, 2018): 12 camadas, 12 cabeças, 110M parâmetros.
- **GPT-3** (OpenAI, 2020): 96 camadas, 96 cabeças, 175B parâmetros.
- **GPT-4** (OpenAI, 2023): Estimado em ∼1.7T parâmetros (Mixture of Experts).
- **Gemini Ultra** (Google, 2024): Arquitetura multimodal (texto + imagem + áudio + vídeo).

✔ Takeaways

- Os **Transformers** abandonaram a leitura sequencial (RNNs) e processam todos os tokens em paralelo, revolucionando a escala de treinamento.
- A **Tokenização** moderna (BPE) usa *subwords* para manter o vocabulário eficiente e lidar com palavras inéditas ou complexas.
- O **Positional Encoding** injeta a noção matemática de ordem na sequência através de funções senoidais, já que a rede processa os tokens simultaneamente.
- O mecanismo de **Self-Attention** permite que as palavras olhem para todas as outras palavras na frase (através das matrizes Query, Key e Value) para compreender contexto e resolver correferências.
- A arquitetura **Multi-Head Attention** captura diversas abstrações e relações linguísticas de forma paralela.

✎ Questões: Autoavaliação

1 **[Direta]** Qual é a diferença fundamental entre as arquiteturas RNN e Transformer em relação a como elas leem o texto, e por que o Transformer é essencial para processar o contexto de longas frases?

2 **[Direta]** Por que algoritmos de tokenização como o BPE (subpalavras) são preferidos nos modelos grandes (LLMs) em oposição à tokenização por palavras completas ou apenas por caracteres?

3 **[Direta]** Defina as funções de **Query (Q)**, **Key (K)** e **Value (V)** dentro do mecanismo de Self-Attention.

4 **[Reflexiva]** Se fôssemos treinar um modelo para entender uma linguagem matemática, como a remoção completa do Positional Encoding afetaria o entendimento do LLM ao ler equações como $A - B = C$ ou $B - A = C$?

5 **[Reflexiva]** No processo de Self-Attention, calcular o produto escalar entre o Query de cada token contra todas as Keys gera um custo computacional quadrático ($O(N^2)$) em relação ao tamanho do texto. Como esse custo limita o processamento prático de livros com 10.000 páginas nas arquiteturas básicas, e quais estratégias podem contornar essa restrição de hardware?

7 Conclusão

A arquitetura Transformer unificou o campo da IA: a mesma estrutura (Tokenização → Positional Encoding → Self-Attention → Feed-Forward) é a base de modelos de texto (GPT, LLaMA), de imagem (Vision Transformer — ViT), de áudio (Whisper) e até de proteínas (AlphaFold). Na próxima aula, veremos como consumir esses modelos pré-treinados de forma prática através do ecossistema **Hugging Face**.

Lab. 13: Inspecionando o Cérebro Transformer

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

⋆ Objetivo do Laboratório

O mecanismo de atenção dos Transformers muitas vezes parece "magia negra" matemática. Nesta atividade, você **abrirá o capô** de um modelo Transformer real (BERT), extrairá as matrizes de atenção, visualizará quais palavras o modelo conecta internamente, e implementará um **extrator de atenção quantitativo** que prova, com números, que o modelo sabe que "ele" se refere a "cachorro" (e não a "rua").

8 Parte 1: O Problema M — Estilo Maratona

Problema M: O Raio-X do Cérebro Artificial

Contexto: A *ExplainAI*, uma startup de IA explicável (XAI), precisa demonstrar a investidores que modelos Transformer **não são caixas-pretas**. O pitch é: "Conseguimos abrir o modelo e mostrar exatamente quais palavras ele conecta para tomar decisões." Sua missão é construir essa demo usando o BERT multilingual, extraindo as matrizes de atenção e provando numericamente as conexões.

Frase de teste (ambígua):

> *"O cachorro não atravessou a rua porque **ele** estava muito cansado."*

A palavra "ele" é ambígua: refere-se a "cachorro" ou a "rua"? Um humano sabe a resposta. O Transformer também — e você vai provar.

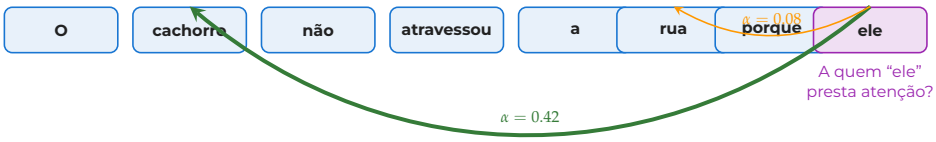
Critérios de Aprovação:

1. Modelo carrega e gera matrizes de atenção com ≥ 12 camadas
2. Tokenização produz a quantidade correta de tokens
3. Na maioria das cabeças de atenção, "ele" atende mais a "cachorro" do que a "rua"

9 Parte 2: Entendendo a Self-Attention

Antes de codar, vamos entender **o que** estamos procurando. O mecanismo de Self-Attention funciona assim: cada token da frase "pergunta" a todos os outros tokens *"quão relevante você é para mim?"*. A resposta é um peso $\alpha_{ij} \in [0, 1]$ — quanto maior, mais o token i "presta atenção" no token j.

O diagrama abaixo ilustra o que esperamos encontrar:



Se o peso de atenção $\alpha_{\text{ele}\to\text{cachorro}}$ for significativamente maior que $\alpha_{\text{ele}\to\text{rua}}$, temos a prova de que o modelo "entendeu" que "ele" se refere ao cachorro. Isso é exatamente o que vamos medir.

• Teoria: Camadas e Cabeças

O BERT tem **12 camadas**, cada uma com **12 cabeças de atenção** — totalizando 144 "lentes" diferentes. Cada cabeça especializa-se em captar um tipo de relação: sujeito-verbo, pronome-referente, adjetivo-substantivo, etc. Nem todas as cabeças captarão a relação "ele → cachorro" — por isso vamos agregar a análise.

10 Parte 3: Configuração e Carregamento (10 min)

A. Instalação

Instale a biblioteca `transformers` do Hugging Face e o PyTorch:

```
pip install transformers torch
```

B. Carregando o Modelo

Crie o arquivo `src/visualizador_attention.py`. Usaremos o BERT multilingual, que suporta português:

```
import torch
from transformers import AutoTokenizer, AutoModel
import numpy as np

# 1. Carrega o BERT multilingual (suporta Portugues)
```

```
nome_modelo = "bert-base-multilingual-cased"
print(f"Carregando {nome_modelo}...")
tokenizer = AutoTokenizer.from_pretrained(nome_modelo)
modelo = AutoModel.from_pretrained(
    nome_modelo, output_attentions=True)
modelo.eval()
```

O parâmetro `output_attentions=True` é crucial: ele instrui o modelo a retornar as matrizes de atenção junto com as saídas normais. Sem ele, as matrizes seriam descartadas internamente. O `modelo.eval()` coloca o modelo em modo de inferência (sem dropout).

C. Tokenizando a Frase

```
# 2. Frase ambigua para analise
texto = ("O cachorro nao atravessou a rua "
         "porque ele estava muito cansado.")

# 3. Tokenizacao
inputs = tokenizer(texto, return_tensors='pt')
tokens = tokenizer.convert_ids_to_tokens(
    inputs['input_ids'][0])

print(f"\nTokens ({len(tokens)}): {tokens}")
```

A lista de tokens será algo como `['[CLS]', 'O', 'ca', '##chor', '##ro', 'na', '##o', ...]`. O BERT usa **WordPiece**: palavras raras são divididas em subpalavras. Localize o token "ele" na lista — você precisará do índice dele para a análise quantitativa.

△ Importante: Tokens ≠ Palavras

"cachorro" pode virar `["ca", "##chor", "##ro"]`. Isso é normal — o modelo divide palavras raras em subpalavras conhecidas. Os tokens especiais `[CLS]` e `[SEP]` são adicionados automaticamente no início e fim da sequência.

11 Parte 4: Extraindo as Matrizes de Atenção (15 min)

A. Forward Pass

Agora passamos a frase pelo modelo e extraímos as matrizes de atenção:

```
# 4. Inferencia (Forward Pass)
with torch.no_grad():
    outputs = modelo(**inputs)

```

```
attentions = outputs.attentions  # Tupla com 12 camadas

print(f"\nCamadas de atencao: {len(attentions)}")
print(f"Shape de cada camada: {attentions[0].shape}")
# [batch=1, heads=12, seq_len, seq_len]
```

O `torch.no_grad()` desativa o cálculo de gradientes, economizando memória. A variável `attentions` é uma tupla com 12 tensores, cada um com shape `[1, 12, S, S]` onde S é o número de tokens. Cada elemento `attentions[camada][0, cabeca, i, j]` é o peso de atenção do token i sobre o token j.

B. Visualizando a Atenção do “ele”

Vamos extrair os pesos de atenção que o token “ele” atribui a cada outro token, usando a média sobre todas as cabeças de uma camada:

```
# 5. Encontrar o indice do token "ele"
idx_ele = tokens.index("ele") if "ele" in tokens else -1
print(f"Token 'ele' encontrado no indice: {idx_ele}")

# 6. Extrair pesos de atencao do "ele" para todos os tokens
# Media sobre todas as cabecas da camada 8 (arbitraria)
pesos_ele = attentions[8][0].mean(dim=0)[idx_ele]

print(f"\n=== ATENCAO DO TOKEN 'ele' (Camada 8) ===")
for i, (tok, peso) in enumerate(zip(tokens, pesos_ele)):
    barra = "#" * int(peso * 50)
    print(f"  {tok:>15} [{peso:.4f}] {barra}")
```

A saída mostrará um “histograma textual”: quanto mais #, mais atenção o token “ele” dá àquela palavra. Você deve ver barras maiores para “cachorro” e menores para “rua”.

12 Parte 5: Prova Quantitativa (10 min)

A. Comparação Direta

Agora vamos ao teste definitivo: agregar os pesos de atenção sobre **todas** as camadas e cabeças, e comparar numericamente:

```
# 7. Funcao auxiliar para encontrar tokens
def encontrar_token(tokens, alvo):
    """Encontra o indice do token alvo."""
    for i, t in enumerate(tokens):
        if alvo in t.lower():
            return i
    return -1

idx_cachorro = encontrar_token(tokens, "cachorro")
```

```python
idx_rua = encontrar_token(tokens, "rua")

# Media sobre TODAS as camadas e cabecas
atencao_total = torch.stack(attentions)  # [12, 1, 12, S, S]
atencao_agregada = atencao_total.mean(dim=[0, 2])[0]

peso_cachorro = atencao_agregada[idx_ele, idx_cachorro].item()
peso_rua = atencao_agregada[idx_ele, idx_rua].item()

print(f"\n=== PROVA QUANTITATIVA ===")
print(f"Atencao 'ele' -> 'cachorro': {peso_cachorro:.4f}")
print(f"Atencao 'ele' -> 'rua':      {peso_rua:.4f}")

if peso_cachorro > peso_rua:
    print("SUCESSO: O Transformer entendeu a referencia!")
else:
    print("NOTA: Pode variar entre camadas/cabecas.")
```

O `torch.stack(attentions)` empilha as 12 camadas em um único tensor 5D. O `.mean(dim=[0, 2])` calcula a média sobre camadas (dim 0) e cabeças (dim 2), resultando em uma única matriz de atenção $S \times S$ agregada.

B. Contagem por Cabeça

Para um argumento ainda mais forte, conte quantas das 144 cabeças (12 camadas $\times$ 12 cabeças) favorecem a conexão correta:

```python
# 8. Contagem: quantas cabecas conectam ele->cachorro?
favoraveis = 0
total_cabecas = 0
for camada in attentions:
    for cabeca in range(camada.shape[1]):
        p_c = camada[0, cabeca, idx_ele, idx_cachorro]
        p_r = camada[0, cabeca, idx_ele, idx_rua]
        total_cabecas += 1
        if p_c > p_r:
            favoraveis += 1

print(f"\nCabecas que conectam 'ele'->'cachorro': "
      f"{favoraveis}/{total_cabecas} "
      f"({favoraveis/total_cabecas:.0%})")
```

Se mais de 50% das cabeças favorecem a conexão correta, temos evidência estatística de que o modelo codificou a referência pronominal. Na prática, espere algo entre 55% e 75% — nem toda cabeça se especializa em referências pronominais.

13 Parte 6: Garantia de Qualidade — Testes CI/CD (5 min)

Crie o arquivo `tests/test_attention.py`:

```python
import torch
from transformers import AutoTokenizer, AutoModel

nome_modelo = "bert-base-multilingual-cased"
tokenizer = AutoTokenizer.from_pretrained(nome_modelo)
modelo = AutoModel.from_pretrained(
    nome_modelo, output_attentions=True)
modelo.eval()

texto = ("O cachorro nao atravessou a rua "
         "porque ele estava muito cansado.")
inputs = tokenizer(texto, return_tensors='pt')
tokens = tokenizer.convert_ids_to_tokens(
    inputs['input_ids'][0])

with torch.no_grad():
    outputs = modelo(**inputs)
attentions = outputs.attentions

def test_modelo_12_camadas():
    """BERT deve ter 12 camadas de atencao."""
    assert len(attentions) >= 12, \
        f"Esperado >= 12 camadas, obteve {len(attentions)}"

def test_tokenizacao_correta():
    """Tokenizacao deve produzir tokens validos."""
    assert len(tokens) > 5, \
        f"Muito poucos tokens: {len(tokens)}"

def test_ele_atende_cachorro():
    """Na media, 'ele' deve atender mais a 'cachorro'
    do que a 'rua'."""
    idx_ele = tokens.index("ele")
    idx_cachorro = next(
        i for i, t in enumerate(tokens) if "cachorro" in t)
    idx_rua = next(
        i for i, t in enumerate(tokens) if "rua" in t)

    att = torch.stack(attentions).mean(dim=[0, 2])[0]
    assert att[idx_ele, idx_cachorro] > att[idx_ele, idx_rua], \
        "Modelo nao conectou 'ele' a 'cachorro'"
```

Execute `pytest tests/test_attention.py -v`. Os **3 testes** devem passar. O terceiro teste é especialmente poderoso: prova computacionalmente que o BERT resolveu a ambiguidade pronominal.

14 Parte 7: Commit e Push

```
git add src/visualizador_attention.py tests/test_attention.py
git commit -m "Lab 13: Self-Attention BERT + prova quantitativa + CI/CD"
git push origin main
```

Conclusão: O que Você Construiu Hoje

✔ Takeaways

- Você **abriu o capô** de um Transformer real (BERT) e extraiu as 12 camadas × 12 cabeças de atenção — 144 "lentes" diferentes que o modelo usa para processar linguagem.
- Visualizou textualmente quais tokens recebem mais atenção do pronome "ele", revelando a **estrutura interna** da compreensão de linguagem.
- Provou **quantitativamente** que o modelo conecta "ele" → "cachorro" (e não "rua"), demonstrando que a Self-Attention resolve referências pronominais.
- Contou quantas cabeças favorecem a conexão correta, mostrando que o conhecimento linguístico é **distribuído** entre múltiplas cabeças.
- Os 3 testes CI/CD garantem: modelo com 12+ camadas, tokenização correta e conexão pronominal verificada.

◇ Informação

Gancho para a Próxima Aula: Você viu como o Transformer processa linguagem por dentro. Mas carregar o BERT, configurar o tokenizer e extrair atenções exigiu dezenas de linhas de código. Na próxima aula, você descobrirá o **Hugging Face Hub** — o "GitHub da IA" — e sua abstração `pipeline()`, que reduz tudo isso a **três linhas de código**. Análise de sentimento, NER, tradução, resumo: tudo com uma única função.