Acesse a versão completa em: https://alessiojr.com/disciplines/ia2/conteudo/00-Livro/

Lab. 13 (Parte 2): Máscaras, Cache KV e Diagnóstico de Atenção

Aléssio Miranda Júnior

Outubro - 2026/2

Lab. 13 (Parte 2): Máscaras, Cache KV e Diagnóstico de Atenção

⋆ Objetivo da atividade

Você distinguirá BERT, GPT e T5 pelo fluxo de informação, resolverá um exercício conceitual de máscaras e cache KV e comparará atenção em duas entradas controladas. A comparação é um **diagnóstico**, não evidência causal nem prova de que o modelo resolveu correferência.

1 Mapa de arquiteturas

Complete a tabela antes de abrir o código.

Família	Blocos	Visibilidade	Uso típico
BERT	encoder	bidirecional	representação/classificação
GPT	decoder	causal: passado e posição atual	geração autoregressiva
T5	encoder–decoder	encoder bidirecional; decoder causal	texto para texto

• Teoria: Máscaras e cache KV

Para uma sequência de quatro posições, desenhe uma máscara causal 4×4: a posição i pode consultar quais colunas j? Em seguida, explique por que, ao gerar o quinto token com GPT, reutilizar as chaves e valores das quatro posições anteriores evita recalcular suas projeções. O cache KV reduz trabalho repetido, mas cresce com o comprimento do contexto e consome memória.

Versão compilada em 11 de novembro de 2026 – Capítulo isolado do livro.

Acesse a versão completa em: https://alessiojr.com/disciplines/ia2/conteudo/00-Livro/

2 Comparação controlada por spans

Use o script da Parte 1. Compare duas frases que só mudam um termo e mantenha fixos modelo, revisão local, tokenização e camada. Execute um *forward pass* por frase e reutilize as atenções retornadas para inspecionar várias cabeças; isso evita repetir o mesmo cálculo para cada medida.

```
def span(word_ids, indice_palavra):
    return [i for i, word_id in enumerate(word_ids)
            if word_id == indice_palavra]

def atencao_entre_spans(matriz, origem, destino):
    # Média das conexões entre todas as subpalavras dos dois spans.
    return matriz[origem][:, destino].mean().item()

def executar_uma_vez(texto_em_palavras):
    entrada = tokenizer(texto_em_palavras, is_split_into_words=True,
                        return_tensors="pt", truncation=True)
    ids = entrada.word_ids(batch_index=0)
    with torch.no_grad():
        resultado = modelo(**entrada)
    return {"palavras": texto_em_palavras, "word_ids": ids,
            "attentions": resultado.attentions}

def medir(resultado, palavra_origem, palavra_destino,
          camada=0, cabeca=0):
    palavras, ids = resultado["palavras"], resultado["word_ids"]
    matriz = resultado["attentions"][camada][0, cabeca]
    return atencao_entre_spans(
        matriz,
        span(ids, palavras.index(palavra_origem)),
        span(ids, palavras.index(palavra_destino)),
    )

base = ["O", "cachorro", "não", "atravessou", "a", "rua",
        "porque", "ele", "estava", "cansado", "."]
controle = ["O", "cachorro", "não", "atravessou", "a", "rua",
            "porque", "ele", "estava", "molhado", "."]

resultados = {nome: executar_uma_vez(frase)
              for nome, frase in [("base", base), ("controle",
                  controle)]}
for nome, resultado in resultados.items():
    print(nome, medir(resultado, "ele", "cachorro"),
          medir(resultado, "ele", "rua"))
```

Versão compilada em 11 de novembro de 2026 – Capítulo isolado do livro.

Acesse a versão completa em: https://alessiojr.com/disciplines/ia2/conteudo/00-Livro/

△ Importante: Interpretação

Registre números, camada, cabeça e as duas entradas. Uma diferença observada pode ser útil para localizar um padrão a investigar, mas não demonstra por que uma previsão ocorreu, qual antecedente o modelo escolheu ou o efeito de uma intervenção. Também não generalize uma única frase para o idioma.

3 Experimento e relatório curto

Repita a extração para três cabeças pré-definidas: $(0,0)$, $(5,3)$ e $(11,7)$, se existirem no modelo carregado. Como os resultados já estão em memória, essa etapa não repete o *forward pass*. Produza uma tabela com quatro colunas: condição, cabeça, atenção `ele` $\rightarrow$ `cachorro` e atenção `ele` $\rightarrow$ `rua`.

Responda no relatório:

1. A máscara causal permitiria a um GPT na posição 3 ler a posição 4? Justifique pela matriz.
2. Qual custo o cache KV troca por velocidade de geração?
3. Que padrão foi observado e qual conclusão causal você **não** pode tirar dele?

4 Fechamento

Teste somente propriedades locais e determinísticas, como a função de span ou a forma de uma máscara criada por você. Não codifique testes que imponham valores de atenção de um modelo remoto.

```python
def mascara_causal(n):
    return torch.tril(torch.ones((n, n), dtype=torch.bool))

def test_mascara_causal_de_quatro_posicoes():
    mascara = mascara_causal(4)
    assert mascara[2, 0] and mascara[2, 2]
    assert not mascara[2, 3]

def test_span_agrupa_subpalavras():
    assert span([None, 0, 1, 1, None], 1) == [2, 3]
```

Versão compilada em 11 de novembro de 2026 – Capítulo isolado do livro.

✔ Takeaways

- BERT, GPT e T5 diferem pela direção de atenção e pelo objetivo arquitetural.
- A máscara causal impede acesso ao futuro; o cache KV reaproveita projeções do passado durante geração.
- Atenção agregada por spans evita erros de subpalavras e serve como diagnóstico controlado, não como prova causal.

Acesse a versão completa em: https://alessiojr.com/disciplines/ia2/conteudo/00-Livro/

Versão compilada em 11 de novembro de 2026 – Capítulo isolado do livro.