

MAE5911/IME: Fundamentos de Estatística e Machine Learning. Prof.: Alexandre Galvão Patriota

Questão 01: Descreva o mecanismo de atenção com múltiplas cabeças (Multi-head Attention). Apresente o desenvolvimento como feito em sala, considerando a versão “mascarada”, para identificar médias ponderadas dinamicamente.

O mecanismo de atenção é o que transforma representação semântica de tokens em representação contextualizada, garantindo que os tokens sejam reinterpretados de acordo com o contexto em que é referido. O mecanismo consiste em durante o treino aprender as matrizes W_Q e W_K que representam parâmetros de *query*, o que este token procura, e *key*, o que este token oferece, tais que

$$Q_i = x_i * W_Q$$

$$K_i = x_i * W_K,$$

no qual, durante o teste, o embedding x de cada token é redefinido para um novo valor utilizando a fórmula

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V = \sum_j A_{ij} V_j$$

na qual M é a máscara que apenas atribui contribuição não nula para tokens em relação a si mesmo e aos tokens predecessores, garantindo uma análise causal de contexto e V é a estatística suficiente sobre a qual tomamos a expectativa, também calculado via matriz de parâmetros W_V , tal que $V_i = x_i * W_V$. O resultado final é a matriz Attention, que tem dimensão $n \times d_{model}$, para n tokens e dimensão do embedding d_{model} , no modelo com uma cabeça.

Para o modelo com multiplas cabeças, o *Multi-head Attention*, a dimensão do embedding é dividida em h cabeças, cada uma com dimensão $d_k = d_{model}/h$. Cada cabeça aprende suas próprias matrizes de parâmetros $W_Q^{(i)}$, $W_K^{(i)}$ e $W_V^{(i)}$, para $i = 1, \dots, h$. O mecanismo de atenção é aplicado separadamente para cada cabeça, resultando em h matrizes de atenção distintas.

Este modelo tem a vantagem de reduzir a dimensionalidade do embedding e permitir a paralelização do cálculo da matriz de atenção, além de permitir que cada cabeça foque em diferentes partes da sequência de entrada x , analisando separadamente diferentes aspectos de relacionamento entre tokens, como por exemplo, semântica, sintaxe, ou interdependências. Essas matrizes são então concatenadas e projetadas de volta para a dimensão original do embedding usando uma matriz de projeção de parâmetros adicional W_O , que combina as matrizes de atenção com pesos diferentes, adicionando uma camada adicional de refinamento do contexto, podendo adicionalmente ser utilizada para reduzir a dimensionalidade do embedding. A fórmula do modelo com multi-head attention é:

$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{Attention}(QW_1^Q, KW_1^K, VW_1^V), \dots, \text{Attention}(QW_h^Q, KW_h^K, VW_h^V)\right)W^O.$$

Exemplo do cálculo de *multi-head attention*, partindo dos seguintes blocos W_Q , W_K e W_V de parâmetros de atenção treinados, e $W_O = I_4$ neste exemplo por simplicidade, para $h = 2$:

Head 1

$$W_Q^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, W_K^{(1)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, W_V^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Head 2

$$W_Q^{(2)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, W_K^{(2)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, W_V^{(2)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Considere um prompt com $n = 3$ tokens:

(Life, is, awesome)

Cada token é representado por um embedding de dimensão $d_{\text{model}} = 4$. Para fins ilustrativos consideramos a seguinte matriz de embeddings:

$$X = \begin{bmatrix} \text{Life} \\ \text{is} \\ \text{awesome} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 4}.$$

Cálculo das projeções Q, K, V para cada token:

$$Q^{(i)} = XW_Q^{(i)}, \quad K^{(i)} = XW_K^{(i)}, \quad V^{(i)} = XW_V^{(i)},$$

Head 1

$$Q^{(1)} = \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \quad K^{(1)} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 3 \end{bmatrix}, \quad V^{(1)} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}$$

Head 2

$$Q^{(2)} = \begin{bmatrix} 0 & 2 \\ 1 & 1 \\ 2 & 1 \end{bmatrix}, \quad K^{(2)} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 3 & 1 \end{bmatrix}, \quad V^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 1 & 2 \end{bmatrix}$$

Calculamos agora, para $d_k = d_{\text{model}}/h = 2$:

$$S^{(i)} = \frac{Q^{(i)}K^{(i)\top}}{\sqrt{d_k}}$$

Head 1

$$S^{(1)} = \frac{1}{\sqrt{2}} \begin{bmatrix} 4 & 2 & 2 \\ 3 & 2 & 4 \\ 4 & 3 & 7 \end{bmatrix}$$

Head 2

$$S^{(2)} = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & 4 & 2 \\ 2 & 3 & 4 \\ 3 & 4 & 7 \end{bmatrix}$$

Aplicamos em seguida a máscara causal, o que impede que o token t seja co-relacionado com tokens em posições subsequentes (futuras) $j > t$, exatamente igual ocorre no treinamento. Os scores mascarados são obtidos por:

$$\tilde{S}^{(i)} = S^{(i)} + M, \quad M = \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{bmatrix},$$

Note que a máscara zera as probabilidades das posições não causais:

$$\tilde{S}_{1,:}^{(1)} = (s_{11}, -\infty, -\infty) \implies \text{softmax}(\tilde{S}_{1,:}^{(1)}) = \frac{(e^{s_{11}}, e^{-\infty}, e^{-\infty})}{e^{s_{11}} + e^{-\infty} + e^{-\infty}} = (1, 0, 0),$$

Aplicando a função softmax linha a linha aos scores mascarados, obtemos as matrizes $A^{(1)}$ e $A^{(2)}$.

Head 1

$$A^{(1)} = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.670 & 0.330 & 0.000 \\ 0.102 & 0.050 & 0.848 \end{bmatrix}$$

Head 2

$$A^{(2)} = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.330 & 0.670 & 0.000 \\ 0.050 & 0.102 & 0.848 \end{bmatrix}$$

A saída da cabeça i é

$$\text{head}_i = A^{(i)}V^{(i)} \in \mathbb{R}^{3 \times 2}.$$

Head 1

$$A^{(1)}V^{(1)} \simeq \begin{bmatrix} 2.000 & 1.000 \\ 1.670 & 1.330 \\ 1.102 & 1.898 \end{bmatrix}$$

Head 2

$$A^{(2)}V^{(2)} \simeq \begin{bmatrix} 1.000 & 1.000 \\ 1.670 & 0.330 \\ 1.102 & 1.747 \end{bmatrix}$$

Considerando, por simplicidade, a projeção de saída como a identidade,

$$W^O = I_4,$$

a saída final do bloco de *multi-head attention* é

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H^{(1)} | H^{(2)}) W_O \in \mathbb{R}^{3 \times 4}.$$

Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \begin{bmatrix} 2.000 & 1.000 & 1.000 & 1.000 \\ 1.670 & 1.330 & 1.670 & 0.330 \\ 1.102 & 1.898 & 1.102 & 1.747 \end{bmatrix} \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 1.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{bmatrix}$$

Resultado final:

Multi-Head Attention (novos *embeddings* contextualizados)

$$\text{Attention}(Q, K, V) = \begin{bmatrix} \text{Life} \\ \text{is} \\ \text{awesome} \end{bmatrix} = \begin{bmatrix} 2.000 & 1.000 & 1.000 & 1.000 \\ 1.670 & 1.330 & 1.670 & 0.330 \\ 1.102 & 1.898 & 1.102 & 1.747 \end{bmatrix}$$

```

1 # -----
2 # Cada camada N da rede neural receberá um módulo de atenção multi-cabeças definido da seguinte forma:
3 #
4 self$MM <- torch::nn_module_list(lapply(1:N_Layers,
5   function(x) torch::nn_multihead_attention(
6     n_embd,           # d_model: dimensão do vetores de embedding
7     N_Head,           # h: número de cabeças
8     dropout = p0,    # técnica para reduzir o overfitting
9     batch_first = TRUE))) # informa a disposição dos dados à biblioteca torch

```

Listing 1: **Definição** dos blocos de multi-head attention treináveis

```

1 # -----
2 # Cria uma matriz TxT, sendo T = número de tokens na sequência de entrada
3 # Garante que será tratado no mesmo device que X (CPU ou GPU) para que eles possam ser operados juntos
4 # Aplica 1 à diagonal superior, transforma 1 em booleano TRUE para que a biblioteca torch saiba quais
5 # posições precisam ser mascaradas
6 #
7 wei <- torch::torch_triu(
8   torch::torch_ones(T, T, device = x$device),
9   diagonal = 1)$to(dtype = torch::torch_bool())

```

Listing 2: **Definição** da Máscara causal

```

1 for (j in 1:self$N) {
2
3   # -----
4   # 1) Pré-normalização (LayerNorm)
5   # Normalização dos embeddings, já que todos embeddings interagirão com todos os outros.
6   # A normalização não altera a informação e estabiliza o treinamento.
7   #

```

```

8 QKV <- self$scale1[[j]](output)
9
10 # -----
11 # 2) Chamada da função Multi-Head Self-Attention mascarada do torch que calcula Q, K e V internamente a
12 # partir de QKV.
13 # -----
14 attn_out <- self$MM[[j]](
15   query      = QKV,      # Input para o cálculo de Q
16   key         = QKV,      # Input para o cálculo de K
17   value       = QKV,      # Input para o cálculo de V
18   attn_mask   = wei,      # Máscara causal
19   need_weights = FALSE  # Dispensa o retorno dos pesos utilizados no cálculo da média ponderada
20 )[[1]]                      # Guarda apenas o resultado da média ponderada, mas não os pesos
21
22 # -----
23 # 3) Conexão residual após atenção
24 # Mantém a informação original dos embeddings e evita que a transformação da atenção distorça
25 # excessivamente a representação.
26 # Estabiliza o fluxo do gradiente.
27 # -----
28 output <- output + attn_out
29
30 # -----
31 # 4) Feed-Forward Network + residual
32 # Aplicação de uma MLP ponto-a-ponto em cada token individualmente, capturando relações não lineares
33 # permitindo que a rede aprenda padrões complexos.
34 # Mantendo o resíduo e aplicando a normalização para estabilizar a operação.
35 # -----
36 output <- output + self$FFN[[j]]( self$scale2[[j]](output) )
37
38 }

```

Listing 3: Cálculo da atenção multi-cabeças mascarada em cada camada

Questão 02: Treine um modelo de linguagem com dimensão embedding de 128, duas camadas e duas cabeças para os dados de Shakespeare.

Descrição do experimento e configuração do modelo

Corpus utilizado

O modelo foi treinado sobre um corpus textual extraído da edição completa das obras de William Shakespeare disponibilizada pelo *Project Gutenberg*, um repositório digital que distribui gratuitamente obras que já se encontram em domínio público, contendo todo o conteúdo produzido pelo autor: peças teatrais, poemas narrativos, sonetos, trechos introdutórios e notas editoriais presentes na edição digital, com aproximadamente 5,4 milhões de caracteres.

THE SONNETS

1

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And, tender churl, mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

2

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tattered weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine.
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.

Figura 1: Trechos do corpus utilizado no experimento, provenientes da obra completa de Shakespeare disponibilizado pelo Projeto Gutenberg.

Vocabulário extraído do corpus

O modelo foi treinado no regime *character-level*, o primeiro passo consistiu em extrair todos os caracteres distintos presentes no arquivo `Shakespeare.txt`. Inclui letras maiúsculas e minúsculas, dígitos, pontuação, símbolos especiais, acentos, quebras de linha e espaços.

O vocabulário final contém **109 tokens**, sendo o primeiro reservado para o símbolo especial `<PAD>` utilizado em operações internas. Segue o conjunto de caracteres identificados:

[1]	"<PAD>"	""	"\t"	"\n"	"\r"	" "	"_"	"_"	"_"	","	";"	";"
[13]	"!"	"?"	".."	"..."	".."	"'"	","	","	"'"	"("	")"	"["
[25]	"*"	"/"	"&"	"#"	"%"	"•"	"\$"	"ø"	"1"	"2"	"3"	
[37]	"4"	"5"	"6"	"7"	"8"	"9"	"a"	"A"	"à"	"À"	"â"	"æ"
[49]	"Ã"	"b"	"B"	"c"	"C"	"ç"	"ç"	"d"	"D"	"e"	"E"	"é"
[61]	"É"	"è"	"ê"	"ë"	"ƒ"	"F"	"g"	"G"	"h"	"H"	"i"	"I"
[73]	"í"	"j"	"J"	"k"	"K"	"l"	"L"	"m"	"M"	"n"	"N"	"ó"
[85]	"0"	"æ"	"p"	"P"	"q"	"Q"	"r"	"R"	"s"	"S"	"t"	"T"
[97]	"™"	"u"	"U"	"v"	"V"	"w"	"W"	"x"	"X"	"y"	"Y"	"z"
[109]	"Z"											

Figura 2: Vocabulário extraído no corpus utilizado (obra completa de Shakespeare).

Hiperparâmetros do modelo

Os hiperparâmetros utilizados no modelo são:

Tabela 1: Hiperparâmetros utilizados no modelo GPT treinado.

Hiperparâmetro	Valor	Descrição
<code>block_size</code>	16	contexto máximo (tokens anteriores).
<code>n_embd</code>	128	dimensão dos embeddings.
<code>N_Layers</code>	2	número de blocos Transformer.
<code>N_Head</code>	2	número de cabeças de atenção.
<code>dropout</code>	0.2	regularização.
<code>learning_rate</code>	0.003	taxa de aprendizado.
<code>batch_size</code>	64	tamanho do mini-batch.
<code>epochs</code>	1200	número total de épocas.

O tamanho de contexto (`block_size = 16`) limita o alcance da atenção, tornando o treinamento mais leve e viável em hardware pessoal. A dimensão dos vetores de embedding (`n_embd = 128`) é significativo, mas combinada ao número reduzido de camadas e a utilização de multiplas cabeças de atenção (`N_Layers = 2, N_Head = 2`), mantém o modelo ágil e compacto, suficiente para capturar regularidades sintáticas e estilísticas do texto. O dropout (0.2) fornece regularização. A taxa de aprendizado escolhida (0.003) foi ajustada empiricamente.

Arquitetura e fluxo do modelo

A arquitetura utilizada neste trabalho segue o padrão dos modelos GPT do tipo *decoder-only*, cujo fluxo computacional está ilustrado na Fig. 3. O processamento inicia-se pela soma entre o embedding dos tokens e o embedding posicional aprendido, resultando em uma representação contínua que preserva a ordem sequencial do texto. Em seguida, cada camada Transformer aplica uma normalização prévia (*LayerNorm*) seguida de um bloco de autoatenção multi-cabeças mascarada, garantindo o comportamento autoregressivo: o modelo só pode utilizar informações de tokens passados para prever o próximo token.

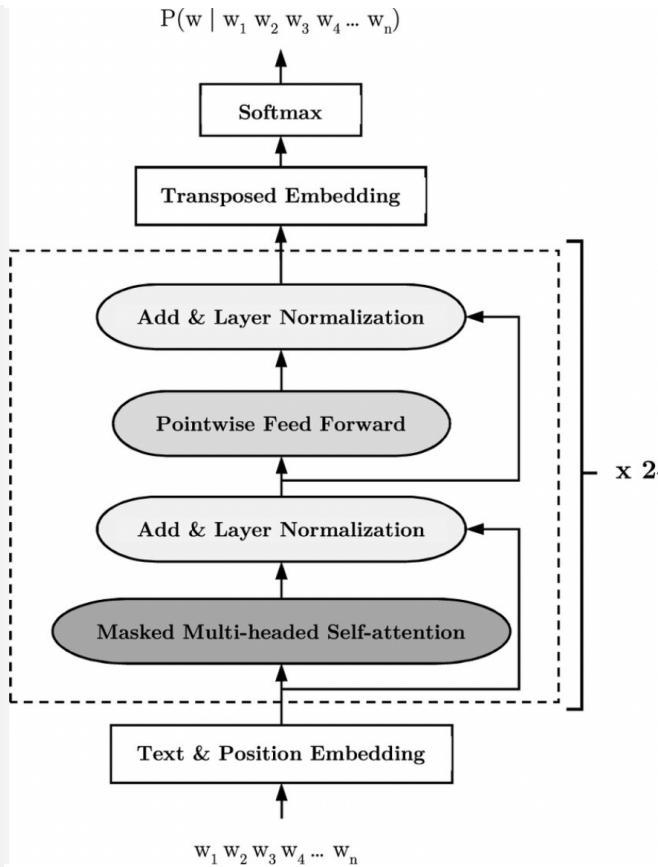


Figura 3: Arquitetura geral do modelo GPT utilizado no experimento.

O resultado da atenção é então somado à entrada original por meio de uma conexão residual, estabilizando o fluxo de gradientes. Após isso, uma nova normalização é aplicada antes do bloco *feed-forward* ponto-a-ponto, responsável por expandir e recomprimir a dimensionalidade interna, permitindo ao modelo capturar não-linearidades locais. Esse bloco também é seguido por uma conexão residual, completando a estrutura *pré-norm* típica de arquiteturas modernas de Transformers. Ao final das camadas empilhadas (2) uma projeção linear — equivalente ao uso do embedding transposto — produz os logits sobre o vocabulário, posteriormente convertidos em probabilidades via *softmax*, que indica o próximo token mais provável.

Treinamento e métricas

Durante o treinamento, foi utilizada a função de perda *cross-entropy*, computada entre os logits produzidos pelo modelo e os caracteres-alvo da sequência. A cada época, registramos tanto a perda de treino quanto a de teste, permitindo acompanhar a evolução do aprendizado e o possível surgimento de sobreajuste.

As curvas de perda apresentaram comportamento regular ao longo das épocas: as perdas de treino e de teste permaneceram próximas, decrescendo de forma suave sem indícios de divergência. Essa proximidade entre as curvas indica ausência de sobreajuste, sugerindo que o modelo conseguiu aprender padrões relevantes do corpus sem memorizar o conjunto de treinamento. Além disso, a estabilidade observada nas últimas épocas reflete um processo de otimização bem-sucedido.

O valor final da perda obtido após 1200 épocas foi de aproximadamente 1.90 para o conjunto de treino e 1.96 para o conjunto de teste.

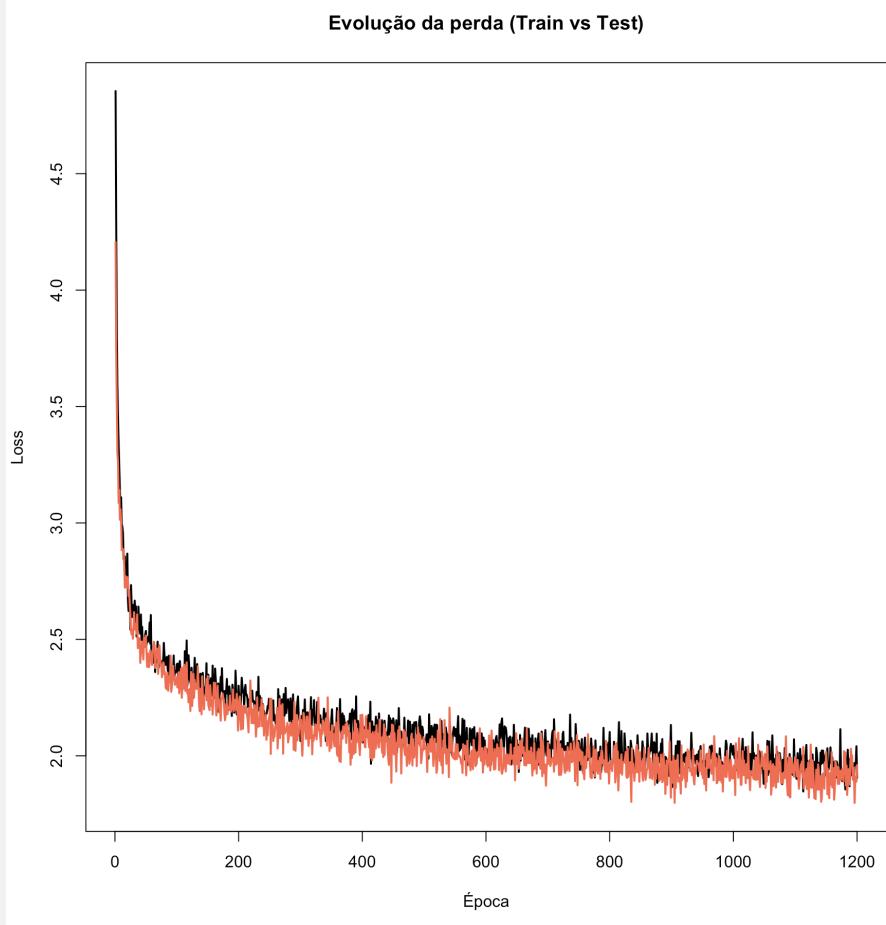


Figura 4: Evolução da função de perda durante o treinamento: comparação entre as curvas de treino e de teste ao longo das 1200 épocas.

0.0.1 Resumo

Em síntese, este experimento demonstra a implementação completa de um modelo Transformer treinado do zero, com pipeline de preparação do corpus, definição dos hiperparâmetros, arquitetura interna e laço de treinamento. Mesmo sendo um modelo compacto, ele é capaz de aprender padrões linguísticos ricos graças à estrutura sofisticada do mecanismo de atenção.

Esperamos observar que, após certo número de épocas, o modelo consiga gerar texto coerente em estilo Shakespeareano, preservando elementos estruturais como métrica aproximada, vocabulário renascentista e padrões sintáticos recorrentes.

Questão 03: Repita o item anterior sem o mecanismo de atenção (com múltiplas cabeças) e descreva o que ocorre no treinamento.
