Arquitetura e Estrutura de Processamento de LLMs

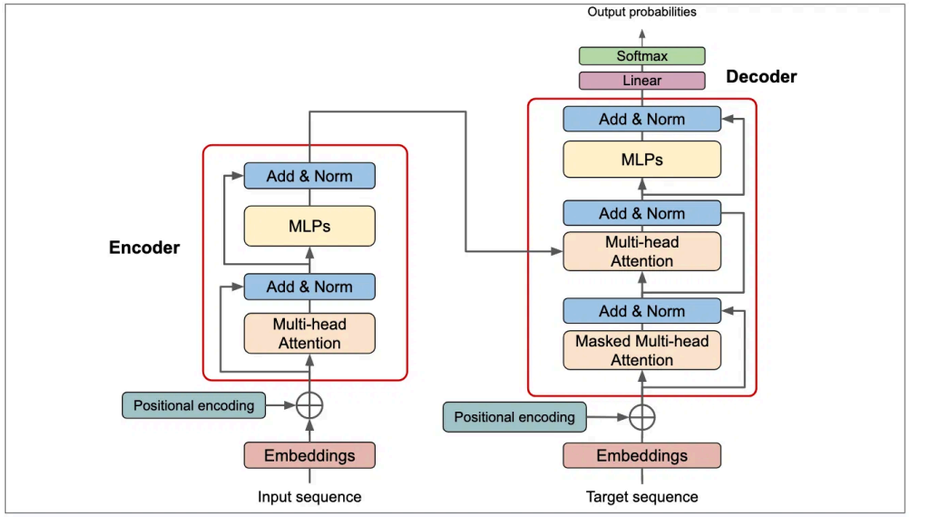
Large Language Models

Uma análise técnica da arquitetura Transformer e dos mecanismos de processamento que impulsionam os modelos de linguagem modernos

Arquitetura Transformer

Fundamentos

A arquitetura Transformer, introduzida em 2017 no artigo "Attention Is All You Need", revolucionou o processamento de linguagem natural ao substituir as redes neurais recorrentes (RNNs) por mecanismos de auto-atenção. Essa mudança permitiu processamento paralelo eficiente e captura de dependências de longo alcance em sequências de texto.



Componentes Principais

O Transformer é composto por dois blocos principais: o **Encoder**, que processa a sequência de entrada e a transforma em representações ricas em contexto, e o **Decoder**, que usa essas representações para gerar a sequência de saída, um token por vez.

> **Vantagens Chave:** Paralelização eficiente durante o treinamento, capacidade de processar sequências longas sem perda de informação, e flexibilidade para adaptar-se a diferentes tarefas de NLP através de fine-tuning.

Variações Modernas

Modelos como BERT utilizam apenas o encoder para tarefas de compreensão, enquanto GPT usa apenas o decoder para geração de texto. Essa modularidade torna o Transformer extremamente versátil.

Mecanismo de Atenção (Self-Attention)

O mecanismo de auto-atenção é o coração do Transformer. Ele permite que cada token na sequência de entrada calcule sua relevância em relação a todos os outros tokens, independentemente da distância entre eles. Isso possibilita a captura de dependências contextuais complexas e de longo alcance.

Fórmula da Atenção

$$\text{Attention}(Q, K, V) = \text{softmax}(Q \times K^T / \sqrt{d_k}) \times V$$

Query (Q)

Representa o token atual que está "perguntando" sobre a relevância de outros tokens. É uma transformação linear do embedding de entrada.

Key (K)

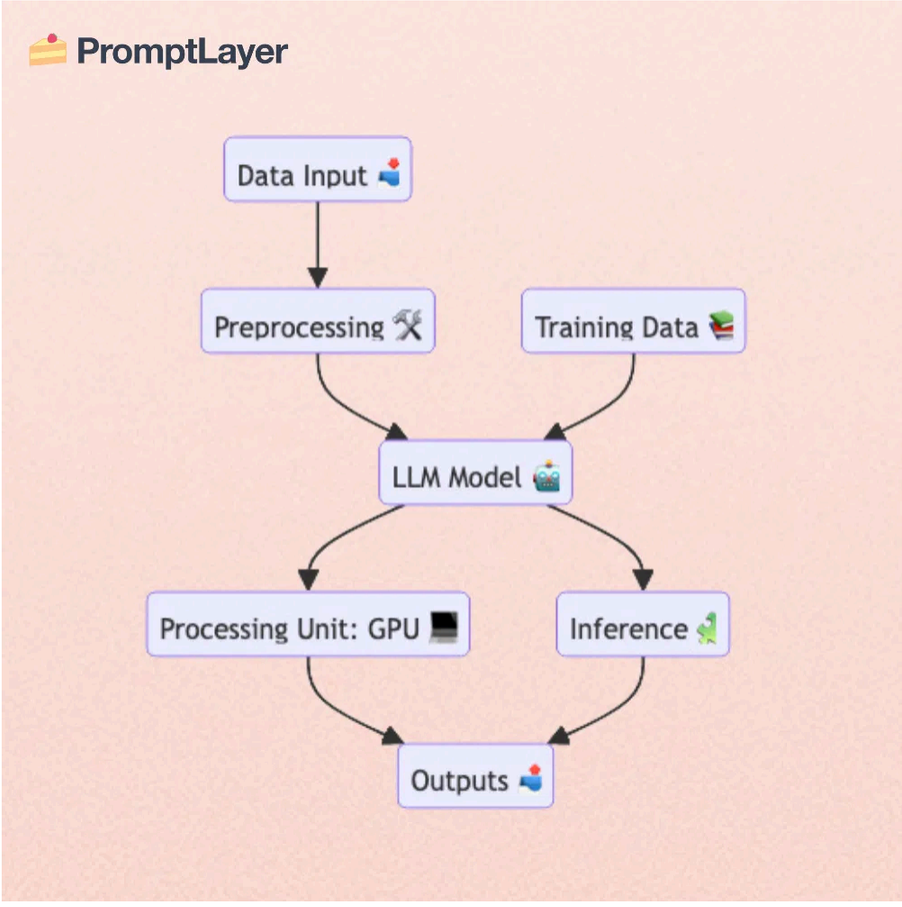
Representa os tokens que estão sendo "consultados". O produto escalar entre Q e K determina a similaridade entre tokens.

Value (V)

Contém a informação real que será agregada. Os valores são ponderados pelos pesos de atenção calculados.

Multi-Head Attention: Os LLMs modernos usam múltiplas "cabeças" de atenção em paralelo. Cada cabeça aprende a focar em diferentes aspectos das relações entre tokens (sintaxe, semântica, dependências de longo alcance), permitindo uma compreensão mais rica e multifacetada do contexto.

Fluxo de Processamento em LLMs



ETAPA 1

Tokenização

O texto de entrada é dividido em unidades menores chamadas tokens. Cada token é mapeado para um ID numérico único do vocabulário.

ETAPA 2

Embedding

Cada ID de token é convertido em um vetor denso de alta dimensão que captura seu significado semântico inicial.

ETAPA 3

Codificação Posicional

Informações sobre a posição de cada token na sequência são adicionadas aos embeddings, permitindo que o modelo entenda a ordem das palavras.

ETAPA 4

Processamento em Camadas

Os embeddings passam por múltiplas camadas de Transformer, onde mecanismos de atenção refinam as representações contextuais.

ETAPA 5

Geração de Saída

O modelo produz probabilidades para o próximo token e gera a saída sequencialmente, um token por vez.

Tokenização - Pseudocódigo

O que é Tokenização?

A tokenização é o primeiro passo no processamento de texto em LLMs. Ela divide o texto de entrada em unidades menores chamadas "tokens", que podem ser palavras, subpalavras ou até caracteres individuais.

Por que é Importante?

Os modelos de aprendizado de máquina processam números, não texto. A tokenização converte cada token em um ID numérico único, permitindo que o modelo processe e aprenda padrões linguísticos.

EXEMPLO

```
Texto: "Hello world"
Tokens: ["Hello", "world"]
IDs: [2534, 8901]
```

Tokens desconhecidos (não presentes no vocabulário) são mapeados para um token especial <UNK>.

```
FUNÇÃO Tokenizar(texto):
    tokens = DividirTextoEmPalavras(texto)
    vocabulario = ObterVocabularioPreDefinido()
    ids_tokens = []

    PARA cada palavra EM tokens:
        SE palavra EXISTE EM vocabulario:
            Adicionar vocabulario[palavra] A ids_tokens
        SENÃO:
            // Token para palavras desconhecidas
            Adicionar vocabulario['<UNK>'] A ids_tokens

    RETORNAR ids_tokens
```

Embedding - Pseudocódigo

Após a tokenização, cada ID de token é convertido em um vetor denso de números reais chamado **embedding**. Esses vetores capturam o significado semântico das palavras em um espaço multidimensional, permitindo que o modelo entenda relações e similaridades entre tokens.

- **Matriz de Embeddings:** Uma tabela pré-treinada onde cada linha corresponde a um token do vocabulário e contém seu vetor de embedding.
- **Dimensionalidade:** Tipicamente entre 256 e 1024 dimensões para LLMs modernos, capturando nuances semânticas complexas.
- **Aprendizado:** Os embeddings são aprendidos durante o treinamento, ajustando-se para que palavras semanticamente similares fiquem próximas no espaço vetorial.

Exemplo Conceitual

Token "gato" → ID 1523 → [0.21, -0.45, 0.89, ...] Token "felino" → ID 2847 → [0.19, -0.42, 0.91, ...] Vetores próximos indicam significados relacionados.

PSEUDOCÓDIGO

```
FUNÇÃO GerarEmbeddings(ids_tokens, matriz_embeddings):
    embeddings = []

    PARA cada id EM ids_tokens:
        // Busca o vetor de embedding correspondente
        vetor_embedding = matriz_embeddings[id]

        // Adiciona à lista de embeddings
        Adicionar vetor_embedding A embeddings

    RETORNAR embeddings
    // Lista de vetores de embedding


// Exemplo de uso:
ids = [1523, 2847, 4521]
embeddings = GerarEmbeddings(ids, matriz_embeddings)
// embeddings agora contém 3 vetores densos
```

Atenção - Pseudocódigo

Implementação simplificada do mecanismo de self-attention

```
FUNÇÃO CalcularAtencao(query, key, value):
    // query, key, value são vetores ou matrizes

    // Passo 1: Calcular similaridade
    pontuacoes_atencao = MultiplicarMatrizes(
        query,
        Transpor(key)
    )

    // Passo 2: Escalar pontuações
    dimensao_key = ObterDimensao(key)
    pontuacoes_escaladas = Dividir(
        pontuacoes_atencao,
        RaizQuadrada(dimensao_key)
    )

    // Passo 3: Normalizar com softmax
    pesos_atencao = Softmax(pontuacoes_escaladas)

    // Passo 4: Aplicar pesos aos valores
    saida_atencao = MultiplicarMatrizes(
        pesos_atencao,
        value
    )

    RETORNAR saida_atencao
```

1 Cálculo de Similaridade

O produto matricial entre query e key transposta determina o quão relevante cada token é para o token atual.

2 Escalonamento

Dividir pela raiz quadrada da dimensão evita que os valores fiquem muito grandes, estabilizando o treinamento.

3 Normalização

A função softmax converte as pontuações em probabilidades que somam 1, criando pesos de atenção.

4 Agregação

Os pesos de atenção são aplicados aos valores, criando uma representação ponderada que captura o contexto relevante.

Geração de Texto - Pseudocódigo

Como um decodificador LLM gera texto prevendo o próximo token com base no contexto

```
FUNÇÃO GerarTexto(modelo, contexto_inicial, max_tokens):
    tokens_gerados = Tokenizar(contexto_inicial)
    PARA i DE 0 ATÉ max_tokens - 1:
        embeddings_contexto = GerarEmbeddings(tokens_gerados, modelo.matriz_embeddings)
        // Passar embeddings pelo modelo (camadas Transformer do decodificador)
        // Obter probabilidades para o próximo token
        probabilidades_proximo_token = modelo.PreverProximoToken(embeddings_contexto)
        proximo_token_id = EscolherTokenComMaiorProbabilidade(probabilidades_proximo_token)
        Adicionar proximo_token_id A tokens_gerados
        SE proximo_token_id FOR token_fim_de_sentenca:
            QUEBRAR
    texto_gerado = ConverterTokensParaTexto(tokens_gerados, modelo.vocabulario_inverso)
    RETORNAR texto_gerado
```

Processo Autoregressivo

Este pseudocódigo ilustra a geração autoregressiva, onde cada token gerado é adicionado ao contexto e usado para prever o próximo token. O modelo processa toda a sequência de tokens gerados até o momento através das camadas do Transformer, produzindo uma distribuição de

Conclusão

Os Large Language Models representam um avanço significativo na inteligência artificial, fundamentados na arquitetura Transformer e em mecanismos sofisticados de processamento de linguagem natural.

01

Arquitetura Transformer

Base dos LLMs modernos, permitindo processamento paralelo e captura de dependências de longo alcance através de mecanismos de atenção.

02

Mecanismo de Atenção

Permite que cada token avalie sua relevância em relação a todos os outros tokens na sequência, criando representações contextuais ricas.

03

Fluxo de Processamento

Desde tokenização até geração de saída, cada etapa transforma o texto em representações progressivamente mais sofisticadas.

04

Geração Autoregressiva

O processo iterativo de prever o próximo token com base no contexto permite a geração coerente de texto longo e complexo.