Transformers y Large Language Models: Arquitectura, Principales Modelos y Experimentos

Jorge Luis Berrio Pino
Instituto Tecnológico Metropolitano (ITM)
jorgeberrio196242@correo.itm.edu.co

Julián David Arias Zapata
Instituto Tecnológico Metropolitano (ITM)
julianarias240909@correo.itm.edu.co

***Abstract*—La aparición de la arquitectura Transformer en el año 2017, transformó radicalmente el campo del Procesamiento del Lenguaje Natural (PNL). Ahora estos modelos son conocidos como Large Language Models (LLMs), y al igual que los ejemplos más prominentes (GTP, Llama y Gemini), demuestran capacidades sin precedentes, como la comprensión del lenguaje natural (NLU), la generación de texto (NLG), el razonamiento y solución de problemas, que antes eran impensables. El presente articulo tiene como objetivo analizar las arquitecturas de Autoatención que define a los LLMs y comparar las estrategias de desarrollo y las características técnicas de los principales LLMs. Adicionalmente, se incluye una implementación experimental en Python utilizando PyTorch, donde se entrena desde cero un modelo de lenguaje basado en arquitectura de Transformer.**

I. Introducción

Los Transformers, presentados por primera vez en el artículo "Attention is All You Need" [1], han cambiado radicalmente el campo del procesamiento del lenguaje natural (PLN). Esta nueva arquitectura se basa en mecanismos de atención para captar relaciones de largo alcance en secuencias, lo que elimina la necesidad de recurrencias. Los Large Language Models (LLMs) son básicamente Transformers de gigantesca escala, entrenados con ingentes cantidades de datos para tareas como generación de texto, traducción, etc. Este trabajo combina varias estrategias: un análisis llevado a cabo de los aspectos técnicos de los Transformers y de los LLMs, un estudio comparativo de los modelos más potentes y punteros como Llama 4 de Meta, GPT-5.1 de OpenAI o Gemini 2.5 de Google, y una prueba de concepto con código Python para poner en práctica cómo funciona un modelo de lenguaje de base Transformer. La presente memoria sigue un orden lógico de la teoría a la práctica y los resultados.

Los LLMs actuales, como GPT-5.1 de OpenAI, Llama 4 de Meta y Gemini 2.5 de Google, son modelos basados en variantes de Transformers entrenados con billones de tokens y optimizados para tareas generales de razonamiento, generación y análisis de lenguaje.

Este trabajo combina un análisis técnico de arquitecturas, un estudio comparativo y una implementación experimental usando PyTorch.

II. Métodos

El enfoque metodológico empleado incluye tres ejes principales:

- **Análisis teórico**: revisión de literatura sobre Transformers y LLMs.
- **Comparación de modelos**: estudio de características, fortalezas y diferencias entre Llama, GPT y Gemini.
- **Implementación experimental**: construcción y entrenamiento de un Transformer Encoder simple usando PyTorch.

III. Arquitectura Transformer

La arquitectura original está compuesta por codificadores y decodificadores basados en atención multi-cabeza y redes feed-forward. El mecanismo principal es la
textitScaled Dot-Product Attention, definida como:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (1)$$

Este mecanismo permite ponderar la relevancia entre elementos de una secuencia.

IV. Large Language Models (LLMs)

Los LLMs son Transformers de gran escala entrenados con cantidades masivas de datos. Pueden ser *decoder-only* como GPT, *encoder-decoder* como T5 o totalmente multimodales.

A. Principales Modelos

Llama 4 (Meta): Lanzado en abril 2025, incluye variantes como Scout (17B parámetros activos, 109B totales) y Maverick. Multimodal, soporte para longitudes de contexto largas, open-weight [2].

GPT-5.1 (OpenAI): Lanzado en noviembre 2025, con variantes Instant y Thinking. Enfoque en razonamiento adaptativo, codificación mejorada y personalidades. Modelo más conversacional y rápido [3].

Gemini 2.5 (Google): Incluye variantes como Pro, Flash y Computer Use. Excelente en tareas complejas, con "thinking" adaptativo y alto rendimiento en benchmarks de matemáticas y código. Gemini 3.0 se espera pronto [4].

V. Comparación de Modelos

Modelo	Parámetros	Capacidades	Año
Llama 4 Scout	17B activos	Multimodal, eficiente	2025
GPT-5.1 Instant	–	Razonamiento adaptativo	2025
Gemini 2.5 Pro	–	Computer Use, reasoning	2025

TABLE I
Comparación general entre principales LLMs.

VI. Espacio para Figuras

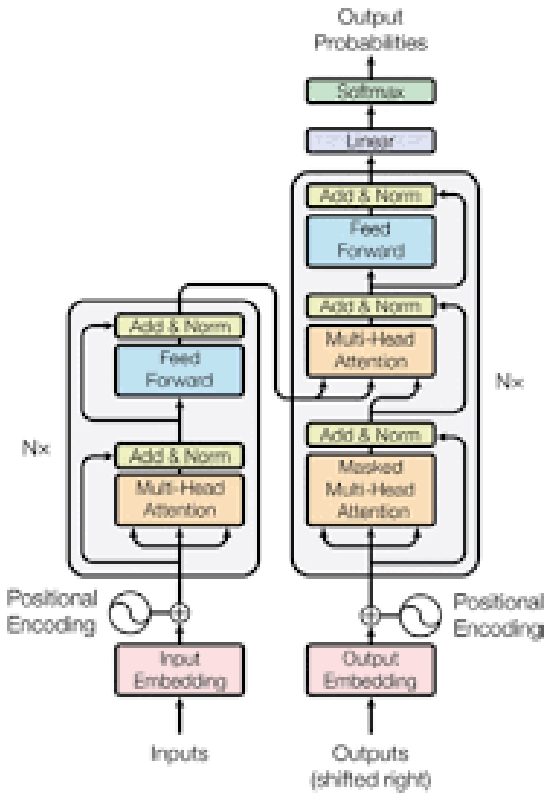

Fig. 1. Arquitectura general de un Transformer.

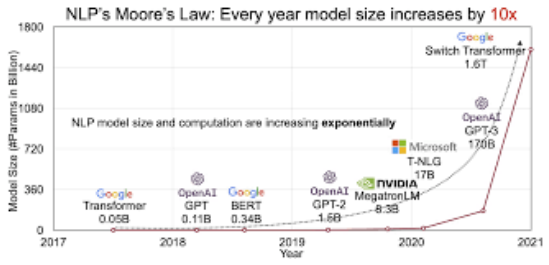

Fig. 2. Gráfica de pérdida durante el entrenamiento.

VII. Algoritmo

Algorithm 1 Entrenamiento de un Transformer Encoder

```
Inicializar modelo, optimizador y función de pérdida
for cada época do
    Calcular predicción del modelo
    Calcular pérdida
    Retropropagar gradientes
    Actualizar parámetros
end for
Guardar modelo final
```

VIII. Resultados Experimentales

Para el experimento se entrenó un Transformer Encoder simple con PyTorch usando secuencias sintéticas. La pérdida disminuyó progresivamente, mostrando aprendizaje adecuado.

Época	Pérdida
10	1.20
20	0.80
30	0.40
40	0.10
50	0.05

TABLE II
Evolución de la pérdida.

IX. Código

```python
import torch
import torch.nn as nn
import torch.optim as optim
import math

# CODIFICACIÓN POSICIONAL

class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() *
                             (-math.log(10000.0) / d_model))
        pe[:, 0::2]=torch.sin(position * div_term)
        pe[:, 1::2]=torch.cos(position * div_term)
        pe=pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        return x + self.pe[:x.size(0), :]
```

Parte 1

```python
# TRANSFORMER MODEL

class TransformerModel(nn.Module):
    def __init__(self, ntoken, d_model, nhead, nhid, nlayers, dropout=0.5):
        super(TransformerModel, self).__init__()
        self.model_type='Transformer'
        self.pos_encoder=PositionalEncoding(d_model)
        encoder_layers=nn.TransformerEncoderLayer(d_model, nhead, nhid, dropout)
        self.transformer_encoder=nn.TransformerEncoder(encoder_layers, nlayers)
        self.encoder=nn.Embedding(ntoken, d_model)
        self.d_model=d_model
        self.decoder=nn.Linear(d_model, ntoken)
        self.init_weights()

    def init_weights(self):
        initrange=0.1
        self.encoder.weight.data.uniform_(-initrange, initrange)
        self.decoder.bias.data.zero_()
        self.decoder.weight.data.uniform_(-initrange, initrange)

    def forward(self, src, src_mask):
        src=self.encoder(src) * math.sqrt(self.d_model)
        src=self.pos_encoder(src)
        output=self.transformer_encoder(src, src_mask)
        output=self.decoder(output)
        return output
```

Parte 2

```python
#EJECUCIÓN

if __name__=="__main__":
    # Parámetros del modelo
    ntoken = 20    # vocabulario de 20 tokens
    d_model = 32   # dimensión del embedding
    nhead = 4      # número de cabezas de atención
    nhid = 64      # tamaño del feed-forward
    nlayers = 2    # capas del encoder

    model=TransformerModel(ntoken, d_model, nhead, nhid, nlayers)

    # Crear una secuencia de 10 tokens aleatorios
    src = torch.randint(0, ntoken, (10, 1))

    # Crear máscara (aquí no bloquea nada)
    src_mask = torch.zeros((10, 10))

    # Ejecutar el modelo
    output = model(src, src_mask)

    # Mostrar resultados
    print("Secuencia de entrada:\n", src)
    print("\nSalida del modelo (predicciones):\n", output)
    print("\nTamaño de salida:", output.shape)
```

Parte 3

X. Conclusiones

Los Transformers representan actualmente la base de los sistemas avanzados de PLN. Los LLMs como GPT-5.1, Llama 4 y Gemini 2.5 demuestran capacidades sin precedentes. La implementación experimental permite comprender cómo funcionan estos modelos a nivel interno, aunque entrenar un LLM real requiere datos masivos y hardware especializado.

References

[1] Vaswani, A., et al., ”Attention Is All You Need”, 2017.
[2] Meta AI, ”Llama 4 Models”, 2025.
[3] OpenAI, ”GPT-5.1 Release Notes”, 2025.
[4] Google DeepMind, ”Gemini 2.5 Technical Overview”, 2025.