Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

1. Tổng quan

- Vấn đề : Các mô hình encoder-only (như BERT) là xương sống cho nhiều tác vụ không sinh ra văn bản (non-generative) như phân loại (classification) và truy xuất thông tin (retrieval), đặc biệt là trong các pipeline RAG. Tuy nhiên, phần lớn các hệ thống hiện tại vẫn dựa trên các mô hình cũ như BERT gốc, với những hạn chế về độ dài ngữ cảnh (thường chỉ 512 tokens), thiết kế kiến trúc chưa tối ưu, dữ liệu huấn luyện lỗi thời và hiệu suất tính toán kém.
- Giải pháp (ModernBERT): Giới thiệu một họ mô hình encoder-only hiện đại, được tối ưu hóa cho cải hiệu năng downstream và hiệu suất suy luận, với độ dài ngữ cảnh native 8192 tokens.

2. Những đóng góp chính

- Kiến trúc hiện đại: Kết hợp nhiều cải tiến đã được chứng minh từ các LLM autoregressive vào kiến trúc encoder:
    - RoPE (Rotary Positional Embeddings): Thay thế positional embeddings tuyệt đối, giúp xử lý ngữ cảnh dài hiệu quả và dễ dàng mở rộng.
    - GeGLU Activation: Thay thế GeLU bằng GeGLU, một biến thể của Gated Linear Units, cho cải thiện hiệu năng ổn định.

    - Pre-Normalization: Sử dụng LayerNorm trước mỗi layer (Pre-Norm) để ổn định quá trình huấn luyện.
    - Alternating Attention: Kết hợp attention toàn cục (global) và local (sliding window) xen kẽ để cân bằng hiệu suất và hiệu quả tính toán trên ngữ cảnh dài.
    - Unpadding: Loại bỏ padding tokens trong cả training và inference, xử lý chuỗi liền mạch để tiết kiệm đáng kể bộ nhớ và tính toán.
    - Flash Attention: Tận dụng Flash Attention 2 & 3 để tối ưu hóa hiệu suất trên GPU.
- Thiết kế Hardware-Aware: Mô hình được thiết kế với kiến trúc "Deep & Narrow" (nhiều lớp hẹp) và các thông số (hidden size, số head, ...) được chọn để tối ưu hóa việc sử dụng tensor cores trên một loạt GPU phổ biến (T4, A10, L4, RTX3090/4090, A100, H100), đảm bảo hiệu suất suy luận nhanh chóng.
- Quy mô dữ liệu và Huấn luyện:
    - Huấn luyện trên 2 nghìn tỷ token, với hỗn hợp dữ liệu hiện đại bao gồm cả code, khắc phục điểm yếu của các encoder trước đó.
    - Sử dụng tokenizer BPE hiện đại (dựa trên OLMo) thay vì tokenizer WordPiece cũ của BERT, cho hiệu quả tốt hơn, đặc biệt trên dữ liệu code.
    - Loại bỏ mục tiêu Next-Sentence Prediction, sử dụng tỷ lệ mask 30% thay vì 15% cho MLM.
    - Sử dụng optimizer StableAdamW và lịch learning rate trapezoidal (Warmup-Stable-Decay) để huấn luyện ổn định.
    - Quy trình mở rộng ngữ cảnh (context length extension) được thực hiện cẩn thận sau khi pre-training cơ bản.
- Đánh giá toàn diện: Đánh giá trên một loạt các tác vụ:
    - NLU (GLUE): ModernBERT-base đánh bại DeBERTaV3-base để trở thành mô hình base mạnh nhất trên GLUE.
    - Retrieval (BEIR): State-of-the-art trên cả single-vector (DPR) và multi-vector (ColBERT) retrieval.

    - Long-Context Retrieval (MLDR): Thể hiện hiệu suất vượt trội, đặc biệt trong cài đặt ColBERT.
    - Code Retrieval (CodeSearchNet, StackOverflowQA): Vượt trội nhờ được pre-train trên dữ liệu code.
- Hiệu quả & hiệu suất:
    - Tốc độ: Xử lý nhanh hơn đáng kể so với các encoder hiện đại khác (GTE-en-MLM, NomicBERT), đặc biệt trên ngữ cảnh dài (nhanh gấp ~2.65-3 lần).
    - Bộ nhớ: Có thể xử lý batch size lớn hơn nhiều so với các mô hình cùng loại, nhờ unpadding và thiết kế hardware-aware.

3. Hạn chế:

- Chỉ tập trung vào Tiếng Anh.
- Có thể kế thừa biases từ dữ liệu web.
- Mục tiêu MLM-only có thể không tối ưu cho mọi tác vụ (so với việc kết hợp MLM và RTD như DeBERTaV3).
- Chưa khám phá scaling theo hướng tham số mô hình lớn hơn.

So sánh ModernBERT với Sentence-BERT (SBERT)

Đặc điểm	SBERT	ModernBERT
Bản chất	Là phương pháp fine-tune các mô hình encoder (BERT) có sẵn.	Một kiến trúc encoder mới được pre-train từ đầu.
Mục tiêu chính	Tối ưu hóa việc **suy luận** (inference) để tạo **câu embedding** cho so sánh ngữ nghĩa (semantic similarity) và tìm kiếm ngữ nghĩa (semantic search).	Tạo ra một **encoder hiện đại, mạnh mẽ và hiệu quả** cho mọi tác vụ của encoder: **classification, retrieval (single & multi-vector), NER**, ...
Kiến trúc cốt lõi	Dựa trên kiến trúc BERT/RoBERTa cũ. **Không thay đổi** kiến trúc gốc.	**Kiến trúc được làm mới toàn diện**: RoPE, GeGLU, Pre-Norm,

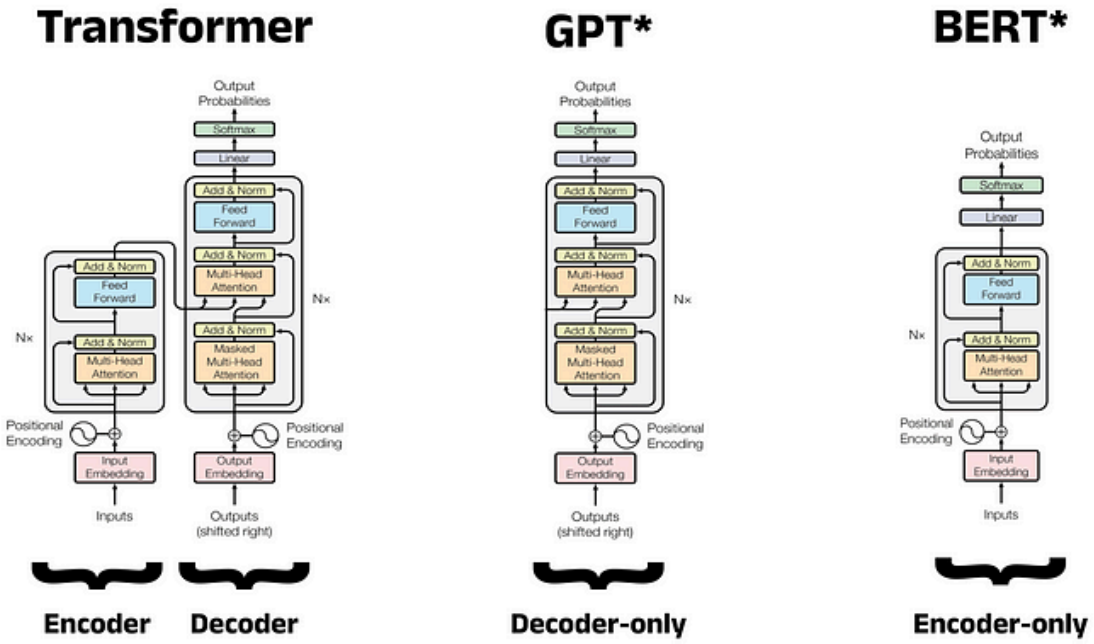
		Alternating Attention, Unpadding.
Độ dài ngữ cảnh	Thừa hưởng giới hạn từ mô hình gốc (thường là **512 tokens**). Gặp khó khăn với văn bản dài.	**8192 tokens** native. Được thiết kế để xử lý ngữ cảnh dài một cách hiệu quả.
Hiệu suất tính toán	Cải thiện tốc độ suy luận so với BERT gốc cho tác vụ similarity bằng cách **tính toán trước embeddings**.	**Hiệu suất vượt trội** nhờ thiết kế hardware-aware, unpadding, Flash Attention. Nhanh hơn các encoder hiện đại khác.
Dữ liệu Huấn luyện	**Fine-tune** trên các dataset song ngữ (e.g., NLI datasets) cho mục tiêu similarity.	**Pre-train từ đầu** trên **2 nghìn tỷ token** dữ liệu đa dạng, bao gồm **code**.
Khả năng Ứng dụng	**Tập trung cao độ** vào **semantic textual similarity** và **semantic search**.	**Đa dạng và toàn diện**: Classification, Retrieval (DPR & ColBERT), NER, Code Understanding, ...
Thế mạnh	- Đơn giản, dễ sử dụng để biến BERT thành sentence encoder. - Hiệu quả tốt cho các tác vụ similarity truyền thống.	**Hiệu năng SOTA** trên nhiều benchmark. **Hiệu suất suy luận cao**. Hỗ trợ **ngữ cảnh dài**. Hiểu **code**. **Linh hoạt** cho nhiều tác vụ.
Điểm yếu	Bị giới hạn bởi các hạn chế của encoder cũ (ngữ cảnh ngắn, kiến trúc kém hiệu quả). Không được pre-train trên dữ liệu mới hoặc code.	Quy mô pre-train rất lớn, khó tái tạo. Hiện chỉ tập trung vào tiếng Anh.

- **BERT gốc (2019)**:
  - Chỉ chạy được context 512 → giới hạn.
  - Chạy khá **chậm** khi sequence dài.
  - Tốn nhiều **VRAM** → khó triển khai trên GPU phổ thông (phải dùng GPU server đắt tiền như A100, V100).
- **ModernBERT (2024):**
  - Dùng nhiều **kỹ thuật tối ưu** như:
    - **Unpadding** → bỏ bớt token rỗng, tiết kiệm tính toán.

- **Alternating Attention (Global + Local)** → không phải mọi token đều "nhìn" hết toàn bộ, giảm chi phí.
- **FlashAttention** → kernel tính toán attention cực nhanh, tiết kiệm RAM.
- **Hardware-aware design** → thiết kế model phù hợp với cách GPU tính toán (tensor cores).

◦ Kết quả:

- **Nhanh hơn 2–3 lần** so với các encoder khác khi xử lý văn bản dài.
- **Ít tốn VRAM hơn** → cùng một GPU, batch size lớn gấp đôi.
- Chạy được trên **GPU phổ thông** mà nhiều công ty/lab/cá nhân đang có.



*Illustrative example, exact model architecture may vary slightly

Architectural Improvements:

Bias terms

- Tắt bias ở hầu hết linear layers + layer norm → tiết kiệm tham số, dùng budget cho phần quan trọng hơn.
- Các LLM gần đây như LLaMA, GPT-NeoX, Gemma, MPT... cũng đã bỏ bias.

  ?) Tại sao phải tắt bias

Positional embeddings

- Dùng **RoPE (Rotary Positional Embeddings)** thay cho absolute positional embedding cũ.
- RoPE tốt cho **long context (8192)** và dễ mở rộng.

Nhắc lại kiến thức:

- **APE (cũ)**: thêm vector vị trí tuyệt đối vào token embedding → đơn giản nhưng bị giới hạn.
- **RoPE (mới)**: xoay (rotate) vector embedding trong không gian 2D theo vị trí của token → encode được khoảng cách tương đối, mở rộng dễ dàng.
- Nói nôn na là : APE ghi số nhà cho từng token (1, 2, 3, ...). Còn RoPE là ghi khoảng cách giữa các nhà (token này cách token kia bao xa.

Normalization

- Dùng **pre-norm** (LayerNorm đặt trước attention/FFN), giúp training ổn định.
- Thêm LayerNorm sau embedding layer.

Activation

- Dùng **GeGLU** (một dạng GLU), thay vì GELU của BERT.
- Được chứng minh cải thiện hiệu quả trong nhiều mô hình mới.
- **Sigmoid** trong GLU bị “kẹt” khi input quá lớn/nhỏ (0 hoặc 1) → gradient dễ biến mất.

Nhắc lại kiến thức:

1. Phần Activation nằm ngay trong **Feed Forward Network (FFN)** của Transformer.

Công thức của FFN:

$$\mathbf{FFN(x)=W2\ f(W1x + b1) + b2}$$

Trong đó:

- f chính là activation function (hàm kích hoạt).
- $W1 \in Rdff \times dmodel$: ma trận "mở rộng chiều" (expansion). Nôm na là map từ vector nhỏ → không gian lớn hơn để xử lý
- $W2 \in Rdmodel \times dff$: ma trận "thu gọn chiều" trở lại.

2. GeLU trong BERT: **BERT gốc** dùng **GELU (Gaussian Error Linear Unit)** thay vì ReLU.

Công thức:

$$\mathbf{GELU(x) = x \cdot \Phi(x)}$$

Trong đó: $\Phi(x)$ là hàm phân phối chuẩn.

Trực quan hơn thì:

- ReLU = "cắt âm, giữ dương".
- GELU = "cho âm qua một ít, giữ dương nhiều hơn" → mượt hơn, gradient ổn định hơn.

3. GLU (Gated Linear Unit)

Sau đó, người ta nghĩ: **tại sao không để mô hình tự học xem thông tin nào cần đi qua, thông tin nào chặn lại?**

Đó là **GLU** (Dauphin et al., 2016):

$$\mathbf{GLU(x) = (W1x) \ \odot \ \sigma(W2x)}$$

Trong đó σ = sigmoid, và $\odot$ = nhân theo phần tử.

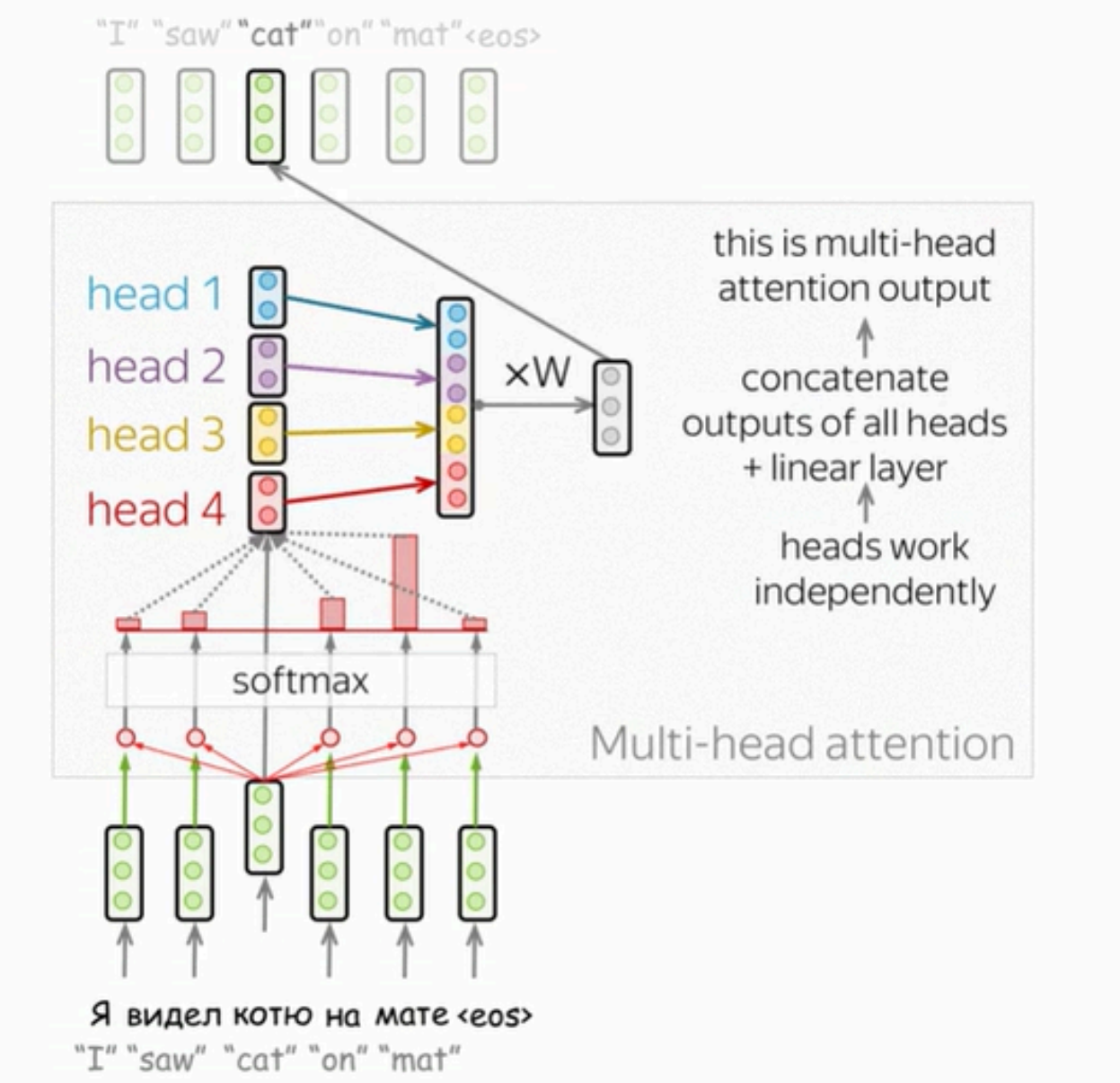
4. GeGLU (Gated GELU)

GeGLU (Shazeer, 2020) = biến thể của GLU, nhưng thay sigmoid bằng **GELU** để làm "cửa" mượt hơn:

$$\mathbf{GeGLU(x) = (W1x) \odot GELU(W2x)}$$

Efficiency Improvements:

1. Alternating Attention:



Nguồn: Seq2seq and Attention

Trong ModernBERT, các lớp global dùng RoPE với θ=160,000 để chịu được ngữ cảnh dài 8k.

Local attention: mỗi token **chỉ** nhìn trong một **cửa sổ trượt 128 token** quanh nó (tiết kiệm tính toán và bộ nhớ vì không còn O(n2)). Các lớp local dùng RoPE với θ=10,000.

ModernBERT **luân phiên** kiểu attention theo **chu kỳ 3 lớp** :

> L1: local → L2: local → L3: global → L4: local → L5: local → L6: global → L7: lo cal →...

→ Tốc độ & bộ nhớ: local attention cắt mạnh chi phí so với global (không còn phải so cặp mọi token). Nhờ "đa số lớp là local", ModernBERT xử lý tài liệu dài nhanh hơn đáng kể; trong đánh giá, họ ghi nhận mức tăng tokens/giây rất lớn ở ngữ cảnh dài **nhờ dùng local attention.**

→ Giảm chi phí tính toán nhưng vẫn giữ khả năng nhìn xa (global).

2. Unpadding

Vấn đề cũ của Transformer:

Câu 1: Tôi ăn cơm. (3 tokens)
Câu 2: Hôm nay trời đẹp. (4 tokens)
Câu 3: Tôi thích học AI lắm. (5 tokens)

→

Câu 1: Tôi ăn cơm [PAD] [PAD]
Câu 2: Hôm nay trời đẹp [PAD]
Câu 3: Tôi thích học AI lắm

Những chỗ [PAD] không có info nhưng vẫn chiếm compute & memory. Và với batch lớn và seq_len dài thì lượng thừa thải sẽ rất lớn.

ModernBERT unpadding:

- Nối lại thành 1 dãy 3 + 4 + 5 = 12 token.
- Sau đó Kernel nhận thêm offsets = [0,5,9,12] để tạo attention block-diagonal

  Mô tả bằng ví dụ sau:

[A A A A A B B B B C C C] → [A A A A A | B B B B | C C C]

[A A A A A .....] ← chuỗi 1 chỉ nhìn trong block 1
[..... B B B B ..] ← chuỗi 2 chỉ nhìn trong block 2
[..... ... C C C] ← chuỗi 3 chỉ nhìn trong block 3

→ Với **local window=128**, mỗi token vẫn chỉ nhìn trái/phải trong cửa sổ **nhưng không bao giờ "vượt biên"** sang block kế bên.

→ Nhờ unpad xuyên suốt mô hình + kernel varlen cho **attention và RoPE**, ModernBERT tránh overhead "unpad/repad lặp lại theo tầng", đo được **+10–20% throughput** so với các cách unpadding trước.

3. Flash Attention:

Vấn đề cũ

- self-attention = tính softmax(QK^T)V.
- Nếu chuỗi dài n phải giữ ma trận QK^T kích thước n×n trong GPU memory → tốn

O(n^2) bộ nhớ. Chuỗi nhiều tokens → GPU nghẹt (out ò memory).

Flash Attention: tính attention theo từng khối (tile), ko lưu toàn bộ ma trận n×n.

Quy trình:

Chia Q, K, V thành block nhỏ → tính QK^t cho block đó → apply softmax → nhân V và tích kết quả → qua block mới → ...

- Dùng **FA-3** cho global attention.
- Dùng **FA-2** cho local sliding window

Nói nôm na là : FlashAttention = "tính attention theo từng miếng nhỏ trong cache, thay vì làm cả đống một lần rồi nghẹt RAM".

Model Design

- Trade-off giữa Deep & Narrow vs Shallow & Wide: ModernBERT chọn Deep & Narrow (nhiều layer, ít hidden size hơn) → tốt cho downstream tasks.
- Thông số:
  - **ModernBERT-base**: 22 layers, 149M params, hidden size 768, GLU exp. 2304.
  - **ModernBERT-large**: 28 layers, 395M params, hidden size 1024, GLU exp. 5248.
- Thiết kế **hardware-aware**: tối ưu để chạy hiệu quả trên GPU phổ biến (3090, 4090, T4, A10, L4...).