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

태그	완료
논문	https://arxiv.org/abs/2412.13663
짧은 소개	디코더 모델의 최신 기술로 인코더 모델 성능 향상

소개

GPT와 같은 대규모 언어 모델(LLM)의 인기가 높아지고 있음에도 불구하고, 인코더 모델은 상대적으로 낮은 연산 요구 사항 덕분에 정보 검색(IR), 문서 처리, 분류 작업 등에 널리 사용되고 있다.

하지만 인코더 모델은 여전히 몇 가지 단점을 안고 있다.

a. 최대 시퀀스 길이가 512 토큰

b. 최신 아키텍처 최적화가 반영되지 않음

c. 최신 도메인 데이터(특히 코드 데이터) 부족

이런 문제를 해결하기 위해 ModernBERT를 만들었다.

a. 최대 8192 토큰 처리.

b. 다운스트림 성능과 효율성을 높이도록 설계된 개선된 아키텍처.

c. 코드 데이터를 포함한 데이터 세트를 사용하여 2조 개의 토큰에 대해 훈련.

방법

1. 아키텍쳐 개선

1.1. Transormer의 최신 발전 사항 통합.

i. Bias Term 제거
최종 decoder linear layer를 제외한 모든 linear layer에서 바이어스를 비활성화.
또한, Layer Norms에서 모든 바이어스를 비활성화.
이를 통해 절약한 메모리를 이용해 linear layer에 더 많은 자원을 투입할 수 있게 됨.

ii. Rotary Positional Embedding
rotary positional embeddings(RoPE) 를 사용하면 모델이 시퀀스 내에서 단어들의 순서뿐만 아니라, 단어의 상대적 위치를 더 잘 이해할 수 있다.
이는 더 긴 컨텍스트를 처리하는 데 유용하며, ModernBERT가 최대 8,192개 토큰의 시퀀스 길이로 확장 가능하게 해줌.

▼ 참조

https://arxiv.org/abs/2104.09864
절대 위치가 아닌 상대 위치 사용.
$q_m^T p_n = \langle f_q(x_m, m), f_p(x_n, n) \rangle = g(x_m, x_n, m-n)$
토큰 임베딩 벡터를 그대로 보존하기 위해, 쿼리와 키에 위치 정보가 반영된 회전 변환을 적용으로써
위치 임베딩이 토큰 임베딩 벡터의 원래 정보를 훼손하는 것을 방지.
따라서 기존과 달리, 매 블록이 시작할 때마다 위치 임베딩을 추가해주므로 단어의 위치 정보를 계속 가져갈 수 있음.

iii. pre-normalization
파라미터가 커질수록 post보다는 pre에 normalization를 적용하는 것이 훈련을 안정화하는 데 도움이 됨.
사전 정규화 블록을 만들어 사용.
임베딩 레이어 뒤에 LayerNorm을 추가하고 반복을 피하기 위해 첫 번째 어텐션 레이어에서 첫 번째 LayerNorm을 제거.

▼ 참조

https://arxiv.org/abs/2002.04745

iv. GeGLU
GeGLU는 원래 BERT의 Activation function인 GeLU보다 향상된 성능을 보임.

$$GeGLU(X) = GELU(XW_1 + b_1) \odot \sigma(XW_2 + b_2)$$
여기서
W_1과 W_2는 각각 다른 가중치이다.
게이트(gating) 메커니즘을 사용하여 특정 정보를 강조하고 불필요한 정보를 억제할 수 있도록 훈련됨.
학습 속도를 높이고 안정성을 더해주며 과적합을 방지해 일반화 성능을 높여준다는 연구 결과들이 존재.

▼ GELU

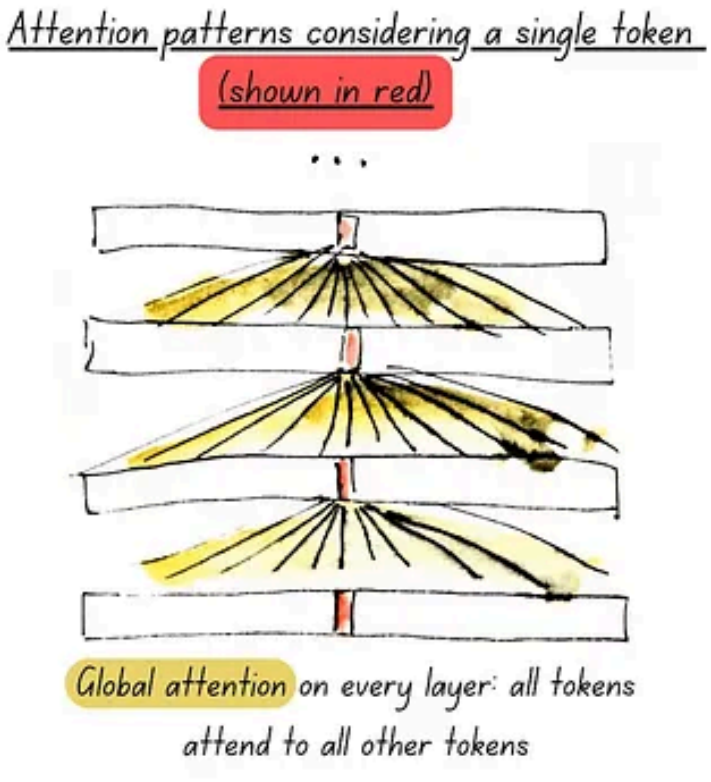
https://arxiv.org/abs/2002.05202

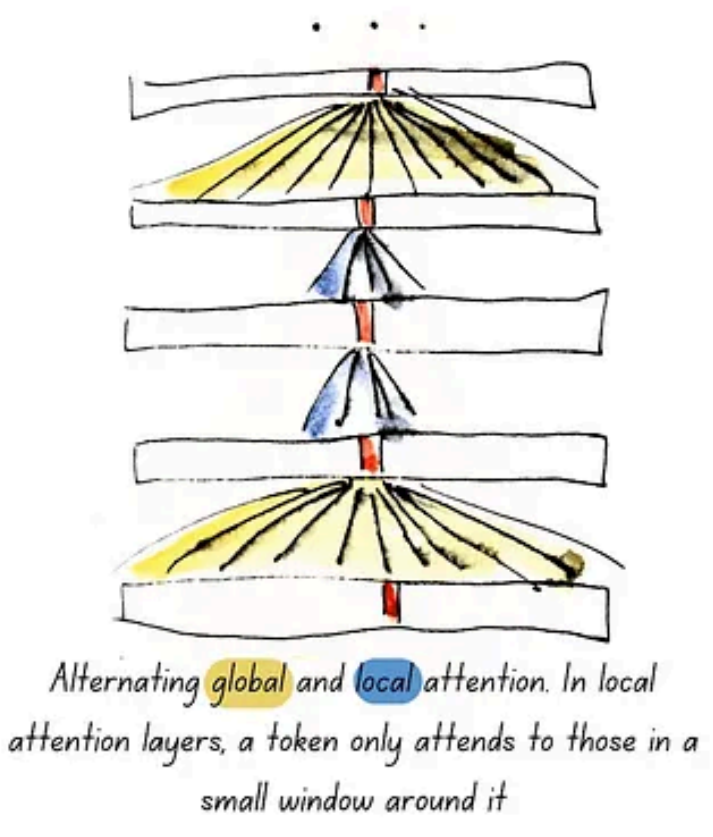
GeLU는 ReLU와 유사하지만, 더 부드러운 비선형성을 제공하는 활성 함수.

$$GeLU(x) = x \cdot \Phi(x)$$
Φ(x)는 표준 정규 분포의 누적 분포 함수(CDF)다.
이는 입력 값을 확률적으로 활성화하는 역할을 하며, 특정 입력에 대해 완전히 차단(ReLU처럼 0으로 변환)하지 않고 연속적인 출력을 유지하는 장점이 있다.

1.2. 효율성 향상

i. Alternating Attention
global attention의 경우 모든 토큰이 다른 모든 토큰에 계산되기 때문에, 문맥이 길어지면 $O(n^2)$으로 계산량이 늘어나는 문제점이 있음.
따라서, 작은 슬라이딩 윈도우 내에서만 토큰이 서로 계산되는 local attention과 global attention을 번갈아 가며 사용.





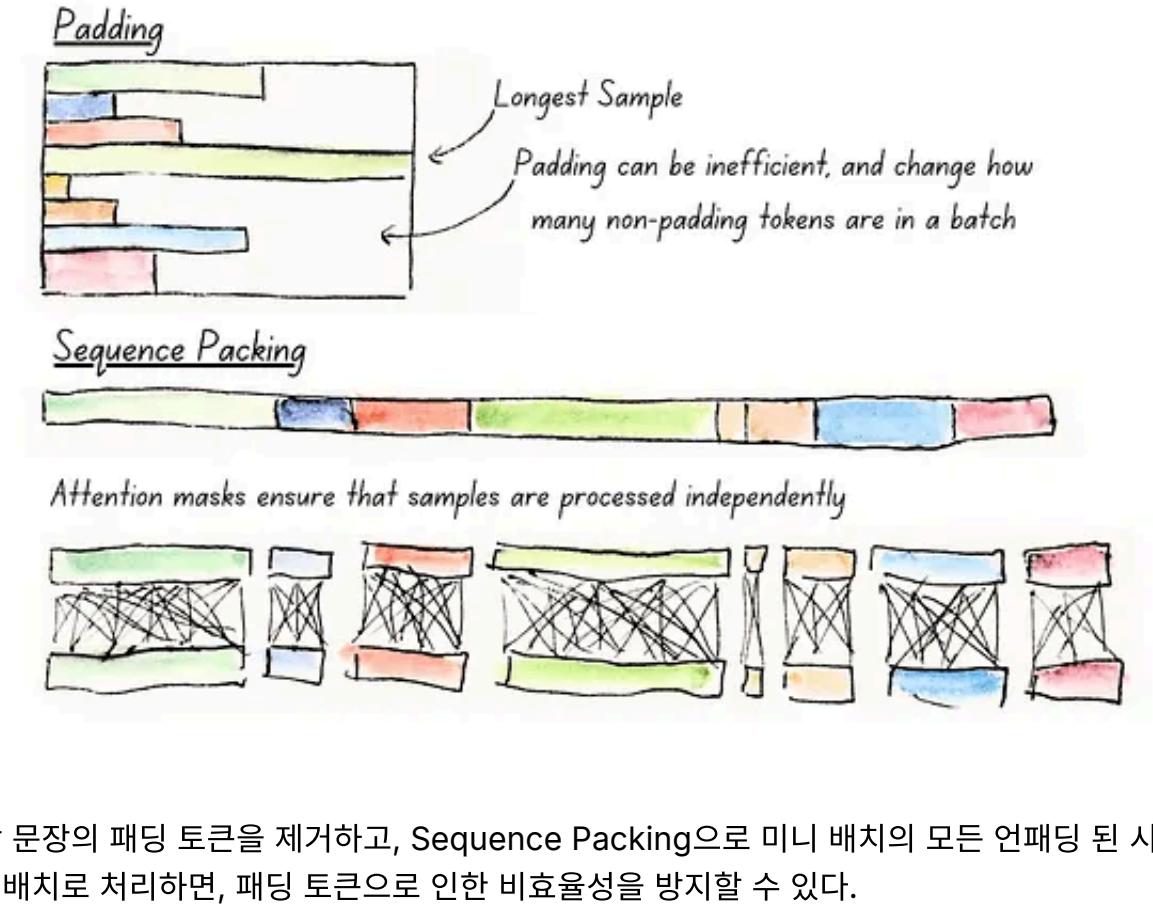
매 세번째 레이어는 모두 16만의 RoPE θ로 전체 토큰을 global attention하고, 나머지 레이어는 1만의 RoPE θ로 128 토큰씩 local attention을 하여 계산량을 줄임.

▼ RoPE θ란?

RoPE 는 위치 정보를 부여할 때 복소수 평면(개념상)에서 각 토큰의 회전 변환(rotation transformation)을 적용하는데, 이때 회전 비율을 조절하는 변수가 θ.

https://devocean.sk.com/blog/techBoardDetail.do?ID=166264&boardType=techBlog

ii. Unpadding & Sequence Packing
인코더 전용 언어 모델은 일반적으로 패딩 토큰을 사용하여 일괄 처리에서 균일한 시퀀스 길이를 보장하지만, 빈 토큰에 컴퓨팅 자원을 낭비.



Unpadding으로 각 문장의 패딩 토큰을 제거하고, Sequence Packing으로 미니 배치의 모든 언패딩 된 시퀀스를 단일 시퀀스로 연결하여 하나의 배치로 처리하면, 패딩 토큰으로 인한 비효율성을 방지할 수 있다.
ModernBERT는 각 문장의 시작 위치와 길이를 저장하는 batch_offsets를 생성.
Flash Attention의 jagged attention mask는 batch_offsets를 사용하여 원래의 각 배치에 해당하는 문장만 slice하여 Attention 계산.
이렇게 필요한 토큰 간에만 attention 연산을 수행하도록 하여, 다른 언패딩 방법에 비해 10-20% 성능 향상. (기존 unpadding 구현에서는 레이어마다 시퀀스를 unpad한 후 다시 pad하는 작업이 반복.)

iii. Flash Attention
글로벌 어텐션 레이어에 플래시 어텐션 3.
로컬 어텐션 레이어에 플래시 어텐션 2. (플레시 어텐션 3에서는 sliding window attention에 대한 지원이 포함되지 않음.)

iv. torch.compile
호환 가능한 모든 모듈을 컴파일하여 교육 효율성을 향상.
이렇게 하면 처리량이 10% 향상되고, 컴파일 오버헤드는 무시할 수 있다.

1.3. 모델 디자인

i. Deep & Narrow 구조
더 깊은 layer 수와 더 좁은 hidden size.
같은 파라미터 수 일 때, Shallow & Wide 구조에 비해 더 나은 다운스트림 성능을 보임.
하지만 layer가 많아지는 만큼 계산해야 할 Activation function이 많아져 추론 속도가 느리다는 단점.

ii. Hardware-aware way Design
ModernBERT는 NVIDIA GPU에 잘 맞도록 디자인되었다.

▼ 디자인 상세

hidden size를 어텐션 헤드로 나누었을 때 64의 배수가 되도록 설정
임베딩 매트릭스를 2의 거듭제곱 또는 64의 배수로 설정
NVIDIA GPU의 Tensor Core에 최적화하여, 가중치 행렬 크기는 64로 나누어 떨어지도록 설정
NVIDIA GPU의 Tile에 맞게 양자화하여, 가중치 행렬은 128 × 256 블록으로 나누어질 수 있도록 설계
블록의 개수가 GPU의 스트리밍 멀티프로세서(SM) 수로 나누어 떨어지도록 하는 것이 목표. (RTX4090은 128개)

- base model : 22 layer / 149M param / 768 hidden / 2304 GLU expansion
- large model : 28 layer / 395M param / 1024 hidden / 5248 GLU expansion

이를 통해 메모리 사용과 연산 부담을 줄이면서도 높은 성능을 유지하여 Deep & Narrow의 추론 속도가 저하되는 것을 최대한 방어했다.

2. 훈련

2.1. Training Data

i. Mixture
웹 문서, 코드 및 과학 문헌을 포함한 다양한 데이터 소스의 주로 영어 데이터로 구성된 2조 개의 토큰에 대해 훈련.

특히 코드 데이터가 들어간 것을 강조함.
mixture rate는 일반적인 최신 조합에 따른다고만 나옴.

ii. Tokenizer
원래 BERT 토크나이저를 재사용하는 대신 최신 BPE 토크나이저를 약간 수정하여 사용.

▼ BPE tokenizer

https://arxiv.org/abs/2402.00838
테스트 결과 이 토크나이저의 성능이 제일 좋게 나왔기 때문에 사용.

- 기존 BERT 모델과의 호환성을 위해 동일한 특수 토큰(e.g., [CLS], [SEP])과 템플릿을 사용.
- 어휘 크기는 50,368로 설정되어 있으며, 이는 64의 배수로 GPU 최적화를 돕고, 향후 응용 프로그램에 사용할 수 있도록 83개의 미사용 토큰을 포함.

iii. Sequence Packing
언패딩의 결과로 미니배치 크기가 제각각이 되는 것을 방지하기 위해 시퀀스 패킹을 채택.
greedy algorithm을 이용하여 99% 이상의 시퀀스 패킹 효율성을 달성하여 배치 크기 균일성을 보장.

▼ 참조

https://arxiv.org/abs/2107.02027
GPU 메모리 블록을 최대한 채우기 위해 Greedy Algorithm을 사용
이때, 각 시퀀스를 배치할 때 최대한 패딩이 적도록 구성하는 것이 목표
- 긴 문장부터 배치하고 남은 공간을 작은 시퀀스로 채움.
- 현재 미니배치에서 남은 공간이 가장 적게 남도록 하는 시퀀스를 선택
- 한 번 배치된 문장은 그대로 유지하며, 다음 미니배치를 채우는 방식

2.2. Training Setting

i. MLM(Masked Language Modeling)
MosaicBERT에서 사용하는 MLM 설정을 따른다.
- Next-Sentence Prediction (NSP)는 성능 향상 없이 오버헤드만 추가하므로 제거.
- 원래 BERT 마스킹 비율인 15%는 최적이 아니므로, 30% 비율을 사용한다.

ii. StableAdamW Optimizer
StableAdamW = AdamW + Adafactor-style update clipping
Adafactor-style update clipping은 Adafactor의 주요 기법 중 하나로, 업데이트 크기가 너무 커지는 것을 방지하기 위해 클리핑(clipping)하는 방법.
특히 큰 모델에서는 Gradient가 매우 커질 수 있고, 이로 인해 과도한 파라미터 업데이트가 발생할 경우 학습이 불안정해질 수 있기 때문.

▼ 참조

https://arxiv.org/abs/1804.04235
Adafactor는 Adam 옵티마이저를 기반으로 한 메모리 효율적인 옵티마이저.
기본적으로 Adam 옵티마이저는 모든 파라미터에 대해 개별적인 2차 모멘트 추정값(제곱 평균)을 저장해야 하므로, 큰 모델에서는 메모리 사용량이 너무 커지는 문제가 발생.
Adafactor는 이 문제를 해결하기 위해 Adam의 두 번째 모멘트를 압축하여 메모리 사용을 줄이는 방식을 사용.

iii. Learning Rate Schedule
trapezoidal Learning Rate schedule은 사다리꼴 모양처럼, 학습 초기에는 학습률을 점진적으로 증가시키고, 중간에는 일정하게 유지, 마지막에는 다시 감소시키는 방식이다.
특히, 큰 모델을 학습할 때 과적합을 방지하고 최적의 수렴을 유도하는 데 효과적.

iv. Batch Size Schedule
Batch size warmup은 중대형 배치 크기로 작업할 때 모델 학습 속도를 높이는 상식적인 트릭.
최적이 아닌 초기 Weight distribution를 업데이트하는 데 전체 배치를 "낭비"하는 대신, 점진적으로 배치 크기를 증가시켜 모델 가중치를 업데이트 함.
large model에서는 500억 토큰을 학습 할 때마다 768배치에서 4,608배치로 점진적으로 증가시킴.

v. Weight Initialization and Tiling
Nvidia Megatron의 초기화 된 가중치를 이용해 ModernBERT-base를 초기화하여 학습.
- 학습 된 768*768 사이즈의 ModernBERT-base 가중치를 가져와서, 1024*1024 사이즈인 ModernBERT-large의 가중치 중앙에 넣고 wraparound한다. (중앙을 기준으로 대칭적으로 값을 배치하는 방법으로 추정.)
- 레이어가 22 → 28로 늘어남에 따라 사이의 레이어들은 특정 간격으로 interpolation 함.
ModernBERT-large의 초기 훈련을 빠르게 만들어줌.

vi. Context Length Extension
1024 시퀀스 길이의 1조 7천억 개의 토큰을 10K RoPE
θ로 1단계 사전학습.
ModernBERT의 네이티브 컨텍스트 길이를 8192 토큰으로 확장하고, 추가로 3,000억 개의 토큰을 160K RoPE θ로 3e-4의 낮은 학습률로 지속적으로 2단계 사전학습.
그 결과, 8k 길이에서도 균형잡힌 성능을 보임.

다운스트림 평가

1. 평가 설정

1.1. 자연어 이해

GLUE(General Language Understanding Evaluation) 벤치마크 활용하여 성능 측정.

				Single Sentence		Paraphrase and Similarity			Natural Language Inference		
	Model	**Params**	**Seq.**	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE
	BERT$^{\beta}$	**110M**	**512**	59.0	93.1	89.5	89.4	91.4	85.4	91.6	78.2
	RoBERTa$^{\alpha}$	**125M**	**512**	63.6	94.8	90.2	91.2	91.9	87.6	92.8	78.7
	DeBERTav3$^{\epsilon}$	**183M**	**512**	**69.2**	95.6	89.5	91.6	**92.4**	**90.0**	**94.0**	83.8
Base	**MosaicBERT-128**$^{\beta}$	**137M**	**128**	58.2	93.5	89.0	90.3	92.0	85.6	91.4	83.0
	NomicBERT-2048$^{\gamma}$	**137M**	**2048**	50.0	93.0	88.0	90.0	92.0	86.0	92.0	82.0
	GTE-en-MLM$^{\delta}$	**137M**	**8192**	57.0	93.4	92.1	90.2	88.8	86.7	91.9	84.8
	ModernBERT	**149M**	**8192**	65.1	**96.0**	**92.2**	**91.8**	92.1	89.1	93.9	**87.4**
	BERT$^{\beta}$	**330M**	**512**	56.2	93.3	87.8	90.6	90.9	86.3	92.8	83.8
	RoBERTa$^{\alpha}$	**355M**	**512**	68.0	96.4	90.9	92.4	92.2	90.2	94.7	86.6
Large	**DeBERTav3**$^{\zeta}$	**434M**	**512**	**75.3**	96.9	92.2	**93.0**	**93.3**	**91.8**	**96.0**	**92.7**
	GTE-en-MLM$^{\delta}$	**434M**	**8192**	60.4	95.1	**93.5**	91.4	89.2	89.2	93.9	88.1
	ModernBERT	**395M**	**8192**	71.4	**97.1**	91.7	92.8	92.7	90.8	95.2	92.1

1.2. 단문 텍스트 검색 - Information Retrieval (IR) [사용처가 가장 많을 것으로 생각되는 작업]

BEIR 벤치마크 활용하여 검색 성능 측정.

MS-MARCO 데이터를 사용하여 모든 기본 모델을 학습

i. 단일 벡터 (문서와 쿼리를 단일 벡터로 변환하여 효율적인 의미 기반 검색 수행)
DPR(Dense Passage Retrieval)방식으로 모델을 학습 및 평가.
- Contrastive Learning으로 미세조정.
- Query와 Passage간의 코사인 유사도를 계산.

▼ DPE(Dense Passage Retrieval)

https://arxiv.org/abs/2004.04906

Open-domain question answering (ODQA) task를 수행하는 방법으로 개발.
적은 수의 question-passage pair만을 가지고 fine-tuning 시키기는 방법.
Contrastive Learning으로 학습.
Query(질문)와 Passage(문서)를 각각 Dense Representation(임베딩 벡터)로 변환하여 벡터 공간에서 가장 유사한 Passage를 검색.
[CLS]를 임베딩으로 활용.

	Model	NFCorpus	SciFact	TREC-Covid	FiQA	ArguAna	Climate-FEVER	DBPedia	FEVER	HotpotQA	MSMARCO	NQ	Quora	SciDocs	Touche2020	CQADupstack	Avg.
Base	BERT	24.3	51.3	49.5	22.8	31.6	21.9	28.2	64.1	47.9	58.5	37.9	83.1	12.9	20.4	28.5	38.9
	RoBERTa	20.4	45.6	52.2	26.1	35.2	22.3	23.1	60.2	45.0	56.0	34.7	84.0	11.4	**21.1**	28.8	37.7
	DeBERTaV3	8.0	22.6	48.4	11.5	26.1	9.7	5.3	17.3	8.0	25.2	12.5	74.7	5.4	14.2	14.2	20.2
	NomicBERT	25.7	52.0	63.0	23.5	35.5	22.9	**30.3**	65.0	48.0	60.6	**42.6**	84.5	12.6	19.0	29.2	41.0
	GTE-en-MLM	**26.3**	54.1	49.7	**30.1**	**35.7**	**24.5**	28.9	**66.5**	**49.9**	**63.1**	41.7	85.2	**14.1**	19.1	32.5	41.4
	ModernBERT	23.7	**57.0**	**72.1**	28.8	35.7	23.6	23.8	59.9	46.1	61.6	39.5	**85.9**	12.5	20.8	**33.1**	**41.6**
Large	BERT	23.3	50.7	48.9	24.0	35.2	22.1	**27.2**	61.7	45.9	59.8	39.5	83.6	13.0	19.5	28.9	38.9
	RoBERTa	23.9	53.4	55.0	33.4	37.6	**23.5**	25.4	65.2	47.1	60.4	43.3	85.8	13.7	21.1	33.0	41.4
	DeBERTaV3	9.6	31.2	56.6	15.8	26.3	14.4	6.8	29.4	15.3	32.4	21.5	79.1	7.0	18.8	19.9	25.6
	GTE-en-MLM	**27.7**	57.6	48.4	**34.0**	35.3	24.0	27.0	**65.4**	**50.8**	64.1	44.9	85.3	**15.6**	21.4	35.5	42.5
	ModernBERT	26.2	**60.4**	**74.1**	33.1	**38.2**	20.5	25.1	62.7	49.2	**64.9**	**45.5**	**86.5**	13.8	**23.1**	**36.5**	**44.0**

Table 7: BEIR Thakur et al. (2021) nDCG@10 scores for single-vector retrieval models.

ii. 다중 벡터 (문서의 모든 토큰을 개별 벡터로 표현하여 정보 손실 최소화)
JaColBERTv2.5 방식으로 모델을 학습 및 평가.
- BGE-M3를 teacher model로 하여 KL-Divergence로 지식 증류한 모델 사용.
- 각 문서는 모든 개별 토큰 벡터로 표시되며 쿼리와 문서 간의 유사성은 MaxSim을 사용하여 계산.

▼ MaxSim

$$MaxSin(Q,P) = \frac{1}{m}\sum_{i=1}^{m} max_{j=1}^{n} similarity(q_i, p_i)$$

각 Query 단어 벡터 q_i에 대해, Passage 벡터 p_j 중 가장 높은 유사도를 선택하고, 이를 평균.

	Model	NFCorpus	SciFact	TREC-Covid	FiQA	ArguAna	Climate-FEVER	DBPedia	FEVER	HotpotQA	MSMARCO	NQ	Quora	SciDocs	Touche2020	CQADupstack	Avg.
Base	BERT	34.2	71.5	69.9	35.0	**49.9**	19.2	42.4	83.1	69.8	45.4	55.4	84.1	14.7	27.0	34.2	49.0
	RoBERTa	33.7	70.8	69.8	37.4	48.9	18.9	39.3	81.2	66.1	43.7	56.3	83.6	14.8	31.7	34.4	48.7
	DeBERTaV3	31.9	68.5	75.5	35.5	46.5	18.3	35.6	78.1	65.3	39.5	50.4	83.7	14.6	31.1	32.3	47.1
	NomicBERT	**35.5**	72.2	73.5	35.9	44.8	19.0	**43.6**	83.9	**71.1**	**46.3**	**58.5**	84.0	15.1	31.3	33.9	49.9
	GTE-en-MLM	35.1	71.5	69.4	36.0	48.5	17.4	41.2	79.9	67.0	44.4	52.8	85.2	15.0	25.4	34.6	48.2
	ModernBERT	35.2	**73.0**	**80.5**	**38.0**	49.1	**22.2**	42.0	**85.8**	70.4	45.4	57.1	**86.3**	**16.0**	**33.9**	**35.1**	**51.3**
Large	BERT	34.6	72.9	68.8	35.5	48.3	19.7	42.4	83.6	70.7	45.9	57.2	84.8	15.2	28.9	34.9	49.5
	RoBERTa	35.0	72.3	74.4	38.7	50.0	19.6	41.0	82.0	66.2	44.7	57.5	85.9	15.3	27.9	**36.0**	49.8
	DeBERTaV3	31.7	70.2	73.3	35.0	46.2	18.0	36.5	79.0	63.2	39.4	51.6	81.1	14.1	28.6	33.1	46.7
	GTE-en-MLM	35.2	72.4	67.2	39.6	50.3	20.8	**44.4**	82.5	72.0	**47.0**	**60.1**	**86.4**	15.9	30.9	35.4	50.7
	ModernBERT	**36.0**	**73.2**	**81.3**	**40.3**	**50.3**	**22.3**	44.1	**85.8**	**72.5**	46.0	59.9	86.1	**16.9**	**34.6**	35.9	**52.4**

Table 8: BEIR Thakur et al. (2021) nDCG@10 scores for multi-vector retrieval models.

1.3. 장문 텍스트 검색

MLDR 벤치마크를 통해 성능 측정

i. 단일 벡터 - Out Of Domain
MS-MARCO(짧은 컨텍스트) 데이터를 사용하여 사전학습.
MLDR(장문 컨텍스트)에 대한 미세조정 없이MLDR로 평가.

ii. 단일 벡터 - In Domain
MS-MARCO(짧은 컨텍스트) 데이터를 사용하여 사전학습 후 MLDR(장문 컨텍스트)에 대한 미세조정 실시.
MLDR로 평가.

iii. 다중 벡터
ColBERT 모델은 MaxSim 때문에 별도의 미세조정 없이 장문에 일반화 가능.
따라서 MLDR에 대한 미세조정 없이 MLDR로 평가.

		IR (DPR)			IR (ColBERT)		NLU	Code	
	Model	BEIR	$MLDR_{OOD}$	$MLDR_{ID}$	BEIR	$MLDR_{OOD}$	GLUE	CSN	SQA
Base	BERT	38.9	23.9	32.2	49.0	28.1	84.7	41.2	59.5
	RoBERTa	37.7	22.9	32.8	48.7	28.2	86.4	44.3	59.6
	DeBERTaV3	20.2	5.4	13.4	47.1	21.9	88.1	17.5	18.6
	NomicBERT	41.0	26.7	30.3	49.9	61.3	84.0	41.6	61.4
	GTE-en-MLM	41.4	**34.3**	**44.4**	48.2	69.3	85.6	44.9	71.4
	ModernBERT	**41.6**	27.4	44.0	**51.3**	**80.2**	**88.4**	**56.4**	**73.6**
Large	BERT	38.9	23.3	31.7	49.5	28.5	85.2	41.6	60.8
	RoBERTa	41.4	22.6	36.1	49.8	28.8	88.9	47.3	68.1
	DeBERTaV3	25.6	7.1	19.2	46.7	23.0	**91.4**	21.2	19.7
	GTE-en-MLM	42.5	**36.4**	**48.9**	50.7	71.3	87.6	40.5	66.9
	ModernBERT	**44.0**	34.3	48.6	**52.4**	**80.4**	90.4	**59.5**	**83.9**

1.4. 코드 검색

CodeSearchNet과 StackOverflow-QA 데이터로 학습.
단일 벡터 검색 작업으로, 모든 모델은 섹션 1.2에서 식별된 최상의 하이퍼 매개 변수를 재사용.
CoIR(CodeIR) 프레임워크를 사용하여 평가.

2. 다운스트림 평가

논문에서는 성능이 향상되었다고 하나 눈에 띄는 향상인지는 의문이다.

i. 자연어 이해
DeBERTa나 GTE와 대동소이

ii. 단문 텍스트 검색
GTE와 대동소이

iii. 장문 텍스트 검색
전반적으로 GTE보다 더 나은 성능을 보임.

iv. 코드
기존에 없던 과제를 추가한 것이기에 월등한 성능 향상을 보임.

효율성

1. 메모리 효율성

   BS(max batch size)가 base-model의 경우 GTE에 비해 세 배 정도 크다.

2. 속도

   TPS(token per second)또한 짧은 컨텍스트 길이에서 초당 14.5-30.9% 더 많은 토큰을 처리하고 긴 컨텍스트 길이에서 98.8-118.8% 더 많은 토큰을 처리하여 GTE-en-MLM보다 더 빠른 속도를 낸다.

			Short			Long	
	Model	Params	BS	Fixed	Variable	BS	Fixed	Variable
Base	BERT	110M	1096	**180.4**	90.2	–	–	–
	RoBERTa	125M	664	179.9	89.9	–	–	–
	DeBERTaV3	183M	236	70.2	35.1	–	–	–
	NomicBERT	137M	588	117.1	58.5	36	46.1	23.1
	GTE-en-MLM	137M	640	123.7	61.8	38	46.8	23.4
	GTE-en-MLM$_{xformers}$	137M	640	122.5	128.6	38	47.5	67.3
	ModernBERT	149M	**1604**	148.1	**147.3**	**98**	**123.7**	**133.8**
Large	BERT	330M	**792**	**54.4**	27.2	–	–	–
	RoBERTa	355M	460	42.0	21.0	–	–	–
	DeBERTaV3	434M	134	24.6	12.3	–	–	–
	GTE-en-MLM	435M	472	38.7	19.3	28	16.2	8.1
	GTE-en-MLM$_{xformers}$	435M	472	38.5	40.4	28	16.5	22.8
	ModernBERT	395M	770	52.3	**52.9**	**48**	**46.8**	**49.8**