Building a Mini Version of BERT from Scratch

Samyakraj Bayar Follow 6 min read · 5 days ago

Introduction

The field of Natural Language Processing has been fundamentally reshaped by transformer-based architectures. Among them, BERT (Bidirectional Encoder Representations from Transformers) stands as one of the most influential breakthroughs in modern AI. Introduced by researchers at Google in 2018, BERT redefined how machines understand language by enabling deep bidirectional context modeling. Instead of reading text sequentially from left to right, BERT processes entire sequences simultaneously, allowing every word to attend to every other word in the sentence.

This shift dramatically improved performance across a wide range of tasks, including question answering, sentiment analysis, and text classification. However, the full-scale BERT model is large and computationally demanding. Training BERT-base requires significant hardware resources, and BERT-large is even more expensive.

For learners and practitioners who want to truly understand how BERT works internally, building a mini version from scratch is an incredibly valuable exercise. It forces you to reconstruct each component — embeddings, attention mechanisms, encoder layers, and training objectives — while keeping the model lightweight and manageable.

In this article, we will walk through the complete conceptual and architectural process of building a mini BERT. The focus is on correctness, clarity, and depth so that you can understand not just how to build it, but why each component exists.

. . .

Understanding the Core Idea Behind BERT

To build a mini version properly, we must first understand what makes BERT unique.

BERT is based entirely on the Transformer encoder architecture introduced in the paper “Attention Is All You Need.” Unlike earlier sequence models such as LSTMs or GRUs, the transformer does not rely on recurrence. Instead, it uses self-attention to model relationships between all tokens in a sequence at once.

The defining characteristic of BERT is bidirectionality. Traditional language models predict the next word in a sequence, which forces them to learn only left-to-right context. BERT, however, masks random words in a sentence and trains the model to predict them using both left and right context. This allows it to build far richer representations of language.

Another important aspect is that BERT is pretrained on large amounts of unlabeled text using self-supervised objectives. After pretraining, it can be fine-tuned on specific downstream tasks with minimal additional architecture changes.

A mini BERT must preserve these foundational principles: transformer encoders, bidirectional attention, and masked language modeling.

. . .

Designing the Architecture of a Mini BERT

The original BERT-base model contains 12 transformer layers, a hidden size of 768, and 12 attention heads, resulting in approximately 110 million parameters. Such a model is impractical for experimentation on modest hardware.

To build a mini version, we reduce the scale while preserving structural integrity. A practical configuration might include two to four transformer layers, a hidden size between 128 and 256, and two to four attention heads. The vocabulary size can also be limited to a few thousand tokens, and the maximum sequence length can be capped at 64 or 128 tokens.

The purpose of scaling down is not to weaken the architecture conceptually, but to make training feasible while retaining all the core components of BERT.

. . .

Input Representation and Embeddings

One of the most important yet often overlooked parts of BERT is how input representations are constructed.

Every input token is represented as the sum of three embeddings: token embeddings, position embeddings, and segment embeddings. Token embeddings represent the meaning of words or subwords. Position

embeddings encode the position of each token in the sequence, since transformers have no inherent notion of order. Segment embeddings distinguish between sentence A and sentence B when training tasks involve sentence pairs.

The final embedding for each token is the element-wise sum of these three components. This combined representation is then passed into the transformer encoder stack.

For a mini BERT, you can implement these embedding matrices as learnable parameters initialized randomly and trained along with the rest of the model.

. . .

The Transformer Encoder Block

At the heart of BERT lies the transformer encoder layer. Each layer consists of two main subcomponents: multi-head self-attention and a position-wise feedforward network.

Self-attention allows each token to compute a weighted combination of all other tokens in the sequence. This is achieved by projecting inputs into three vectors: queries, keys, and values. Attention scores are computed using scaled dot-product attention, where the query of a token is compared with

the keys of all tokens. The result is normalized using a softmax function and used to weight the value vectors.

Multi-head attention means this process happens multiple times in parallel with different learned projections. Each head learns to capture different linguistic relationships such as syntactic dependencies or semantic similarity.

After self-attention, the output passes through a feedforward network, typically consisting of two linear transformations with a ReLU activation in between.

Each sublayer is wrapped with residual connections and layer normalization. Residual connections help gradients flow more effectively during training, while layer normalization stabilizes the distribution of activations.

Stacking multiple encoder layers allows the model to progressively refine contextual representations.

. . .

Pretraining Objectives in Mini BERT

BERT's strength comes largely from its pretraining objectives. To faithfully replicate BERT in mini form, we must implement these correctly.

The first objective is Masked Language Modeling. During training, approximately 15 percent of tokens in each sequence are selected for prediction. Of those selected tokens, 80 percent are replaced with a special [MASK] token, 10 percent are replaced with a random token, and 10 percent are left unchanged. The model is then trained to predict the original token at those positions.

This strategy prevents the model from relying too heavily on the artificial [MASK] token and encourages it to learn genuine contextual representations.

The second objective in the original BERT is Next Sentence Prediction. In this task, the model receives two sentences and predicts whether the second sentence logically follows the first in the original text. Half of the training pairs are true consecutive sentences, and half are randomly paired.

Although later research suggested NSP may not be strictly necessary, including it in a mini BERT helps replicate the original architecture

faithfully.

The total loss during training is the sum of the masked language modeling loss and the next sentence prediction loss.

. . .

Training the Mini Model

Training a mini BERT requires a text corpus, even if it is small. Wikipedia text, book datasets, or other open corpora can be used. The dataset must be tokenized and converted into numerical input IDs.

During each training iteration, masked tokens are generated dynamically. Sentence pairs are also constructed for the NSP task. The input is passed through the embedding layer and transformer stack, and predictions are made for masked tokens and sentence classification.

The optimizer commonly used is Adam or AdamW, with a learning rate around 1e-4 for small models. Because the model is smaller, training can often be done on a single GPU or even a high-performance CPU, depending on the configuration.

Even though the dataset and architecture are smaller, the training process mirrors that of full-scale BERT.

. . .

Evaluating the Mini BERT

After pretraining, the model can be evaluated in several ways. One simple evaluation method is testing masked token prediction on unseen sentences. If the model predicts reasonable words in context, it has learned meaningful representations.

The model can also be fine-tuned for simple downstream tasks such as sentiment classification or text categorization. Fine-tuning typically involves adding a classification layer on top of the representation of the [CLS] token and training on labeled data.

While performance will not match large pretrained models, the goal here is conceptual understanding rather than state-of-the-art accuracy.

. . .

Limitations of a Mini BERT

A smaller model has limited capacity. With fewer layers and parameters, it cannot capture as many nuanced linguistic patterns as the full model. The vocabulary is smaller, training data is limited, and generalization ability is reduced.

However, these limitations are acceptable for educational and experimental purposes. The structural principles remain intact, and the learning experience is significantly enhanced.

. . .

Conclusion

Building a mini version of BERT from scratch is one of the most powerful ways to deeply understand modern NLP systems. It transforms abstract research papers into tangible engineering decisions. Instead of treating BERT as a black box, you begin to see how embeddings interact with attention mechanisms, how pretraining objectives shape representations, and how architectural components cooperate to produce contextual understanding.

The exercise reveals why BERT was such a breakthrough. Its power does not come from any single trick, but from the elegant integration of transformer encoders, bidirectional masking, and large-scale pretraining. Even when scaled down, these principles remain transformative.

More importantly, constructing a mini BERT builds intuition. It prepares you to understand more advanced architectures such as RoBERTa, ALBERT, and other transformer variants. It equips you with the confidence to modify

architectures, experiment with objectives, and innovate beyond existing models.

In a field evolving as rapidly as NLP, mastering the fundamentals is essential. A mini BERT is more than a simplified implementation — it is a foundational step toward truly understanding how machines learn language.

Naturallanguageprocessing AI ML Computer Science

Written by Samyakraj Bayar

1 follower · 1 following

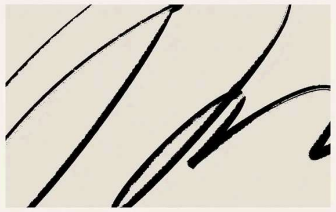
Follow

No responses yet

Write a response

What are your thoughts?

More from Samyakraj Bayar

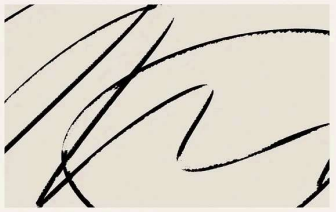


Samyakraj Bayar

RNN vs LSTM vs Transformers: A Deep Dive into Sequence Modeling

Introduction

4d ago 1



Samyakraj Bayar

Fine-Tuning a Model Using Hugging Face Transformers: A...

Introduction

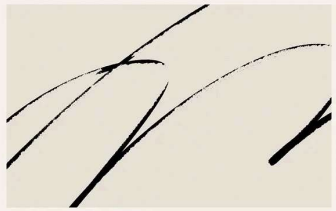
3d ago

Samyakraj Bayar

How GPT-4 Works: A Comprehensive Architecture...

Introduction

3d ago



Samyakraj Bayar

Is Open Source AI the Future?

Introduction

2d ago

See all from Samyakraj Bayar

Recommended from Medium

In Level Up Coding by Christian Bernecker

Building an AI Agent from Scratch with pure Python

From Theory to Implementation: Building a Robust, Self-Correcting AI Agent from...

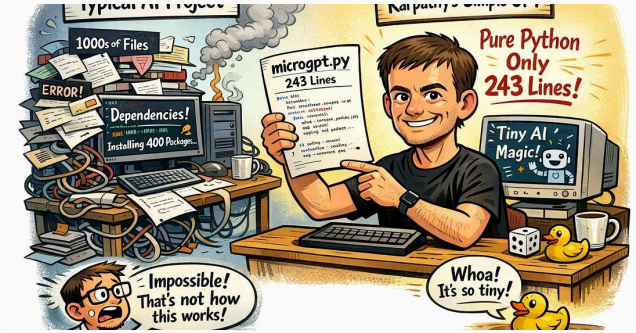
Feb 16 650 6

In CodeX by MayhemCode

Why Thousands Are Buying Mac Minis to Escape Issues with Big...

Something strange happened in early 2026. Apple stores started running low on Mac...

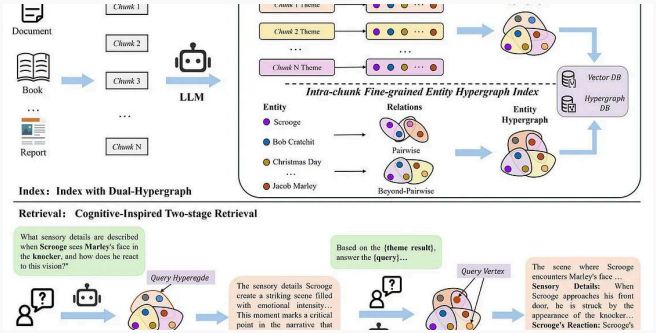
Feb 15 4.2K 69



In Towards Deep Learning by Sumit Pandey

Andrej Karpathy Just Built an Entire GPT in 243 Lines of Python

No PyTorch. No TensorFlow. Just pure Python and basic math.



In Towards AI by Florian June

Cog-RAG: Giving RAG a Brain That Thinks Before It Retrieves

Retrieval-Augmented Generation (RAG) is now a standard way to help LLMs stay...

Feb 16 2.8K 43

Feb 17 254 1

In Data Science Collective by Marina Wyss

Should You Still Learn to Code in 2026?

The answer isn't as obvious as I used to believe.

Feb 23 1.2K 45

In Python in Plain English by Babar saad

8 Python Libraries That Made Me Quit Spreadsheets for Good: Why ...

I didn't abandon spreadsheets because they're outdated. I abandoned them becaus...

Feb 25 133

See more recommendations