MiniGPT: A Minimal Transformer Language Model

FULL TECHNICAL DOCUMENTATION

1. Overview

MiniGPT is a fully functional transformer-based language model implemented from scratch in Python. It demonstrates all core components of a modern transformer, including:

- Tokenization (BPE)
- Embeddings
- Positional Encoding
- Multi-Head Attention
- Feed-Forward Network (FFN)
- Layer Normalization
- Transformer Blocks
- Training Loop
- Text Generation

This documentation includes:

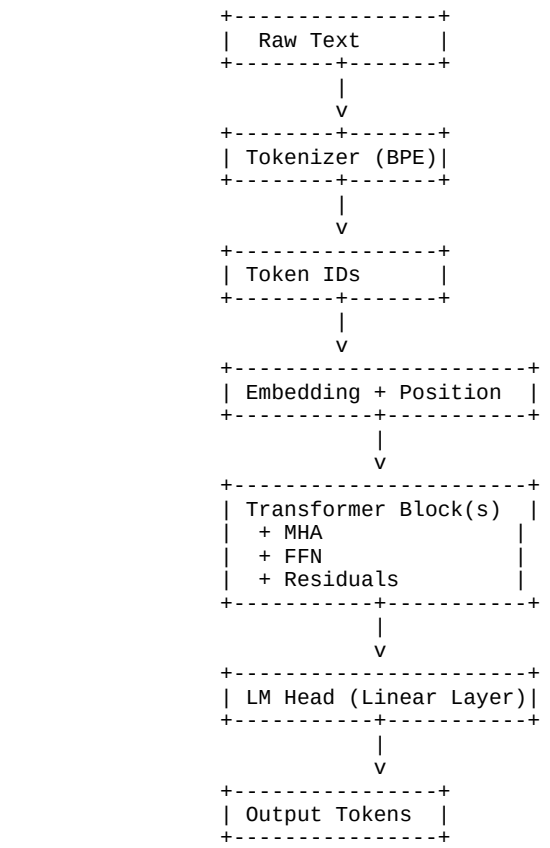
- Architectural overview
- Flowcharts
- Detailed class explanations
- Data flow diagrams
- Step-by-step implementation plan

2. System Architecture

The MiniGPT architecture consists of:

INPUT TEXT → TOKENIZER → EMBEDDING → TRANSFORMER BLOCKS (N LAYERS) → LM HEAD → OUTPUT TOKENS

Flowchart:



3. Detailed Components

3.1 Tokenizer (BPE)

Purpose:
- Convert raw text → tokens using byte pair encoding.

Core Responsibilities:
- Build vocabulary
- Train merge rules
- Encode strings into token IDs
- Decode token IDs back to text

Classes:
- BPETokenizer
  - train()
  - build_vocab()
  - encode()
  - decode()

3.2 Embedding Layer

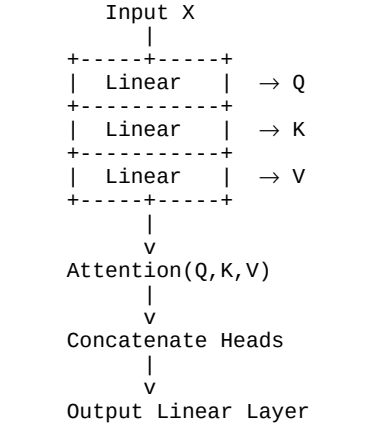
Purpose:
- Convert token IDs into dense vectors
- Add positional encoding

Classes:
- EmbeddingLayer
  - token_embedding
  - position_embedding
  - forward()

3.3 Multi-Head Self Attention

Purpose:
- Each token attends to all previous tokens

Detailed flow:



Classes:
- AttentionHead
- MultiHeadAttention

3.4 Feed Forward Network (FFN)

Purpose:
- Two-layer MLP applied per token

Classes:

- FeedForward

3.5 Layer Normalization & Residuals

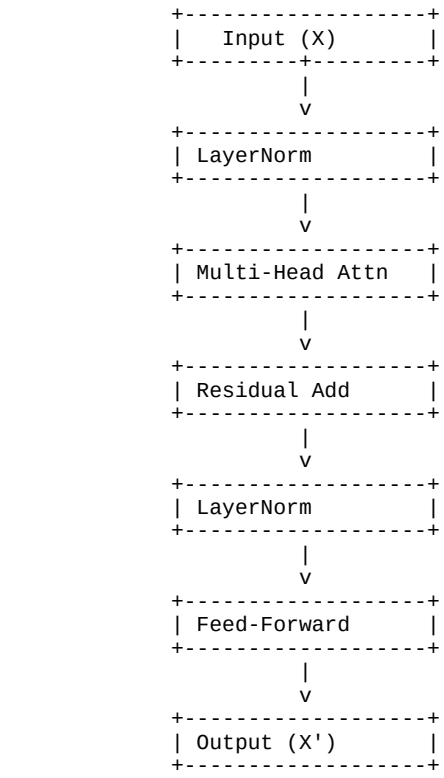
Classes:
- LayerNorm

Every transformer block contains the pattern:
    x = x + MHA(x)
    x = x + FFN(x)

4. Transformer Block

A single block contains:

Flowchart:



Class:
- TransformerBlock
  - forward()

5. Training Procedure

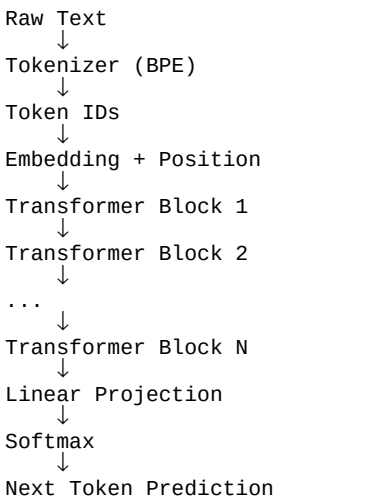
Steps:
1. Load dataset
2. Train tokenizer
3. Convert text → token IDs
4. Build batches
5. Forward pass
6. Compute loss
7. Backprop
8. Update weights
9. Save model
6. Text Generation

Algorithm:
1. Start with prompt tokens
2. Repeatedly:
   - Feed tokens into model
   - Get logits of next token
   - Sample / greedy select next token
   - Append to sequence

7. Full Project Implementation Steps

1. Implement Tokenizer (BPE)
2. Create DataLoader
3. Implement all math helpers (matmul, softmax, etc.)
4. Implement Embedding Layer
5. Implement Positional Encoding
6. Implement Attention Head
7. Implement Multi-head Attention
8. Implement FFN
9. Implement Transformer Block
10. Implement Transformer Model
11. Implement Training Loop
12. Implement Text Generation

8. Flowchart: Final End-to-End Pipeline



9. Future Improvements

- Add dropout
- Add weight initialization strategies
- Add optimizers (AdamW)
- Add GPT-style decoding (top-k, top-p)
- Add masking support
- Add training metrics

END OF DOCUMENT