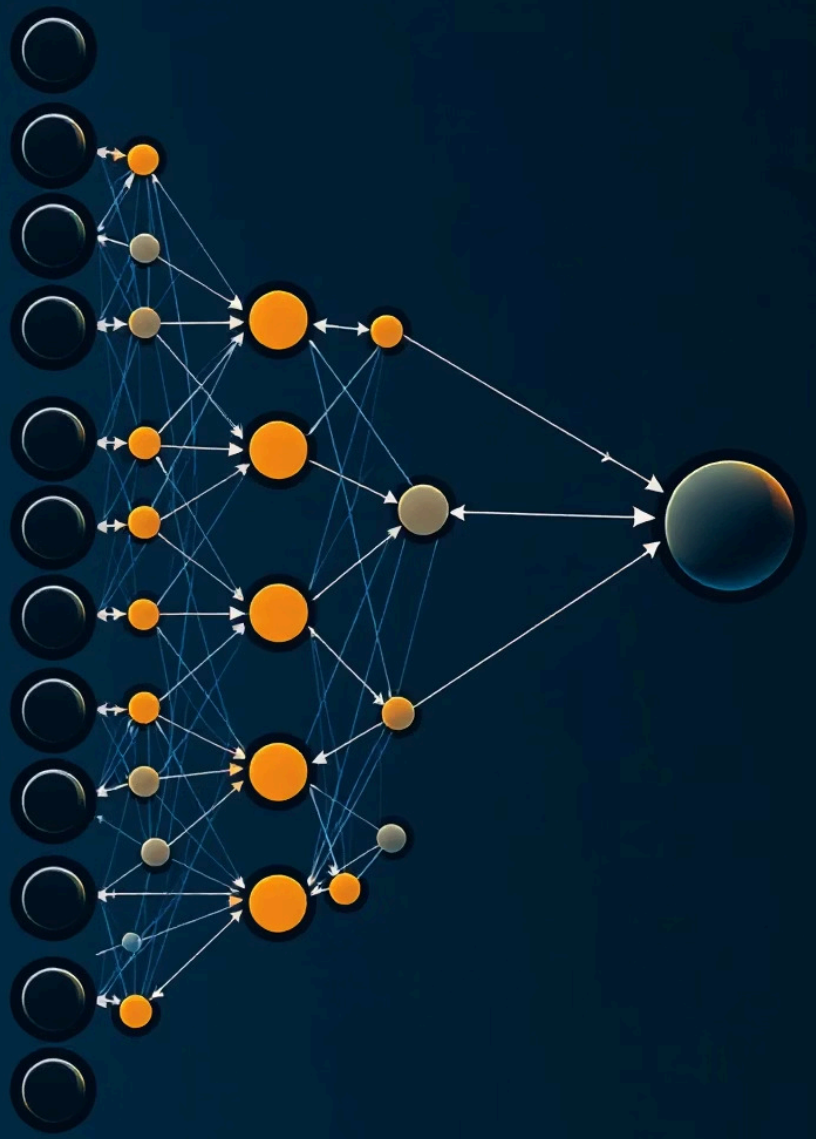
HeartBERT:

BERT-based Model for Sentiment Analysis

Dayeon Hwang / 20231424



Project Overview

Model Implementation

Custom BERT from scratch

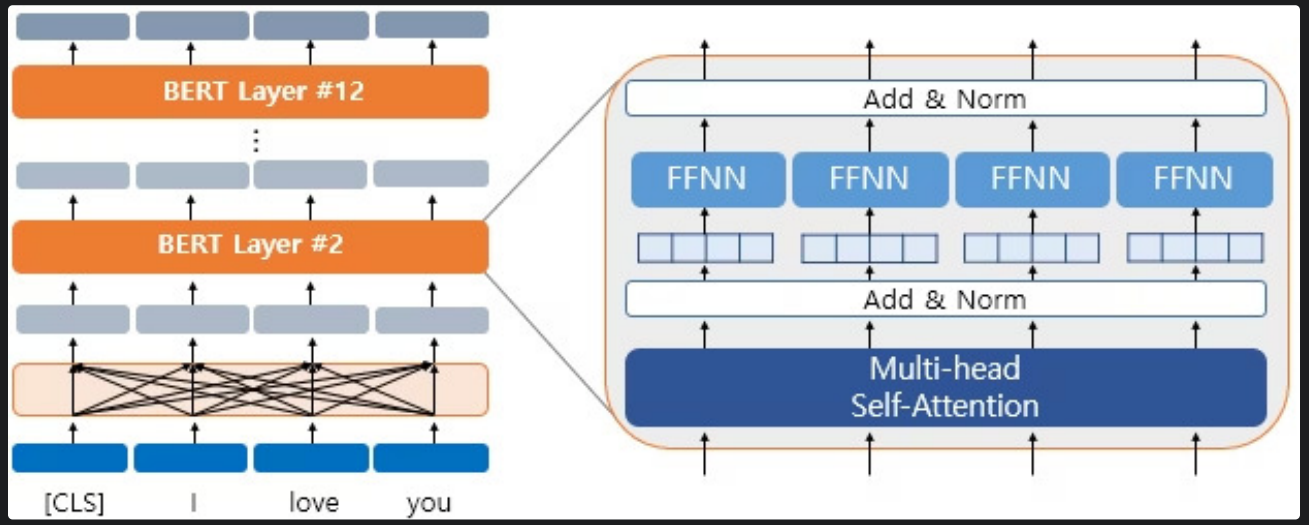
Pretraining

Masked Language Modeling on unlabeled corpus

Finetuning

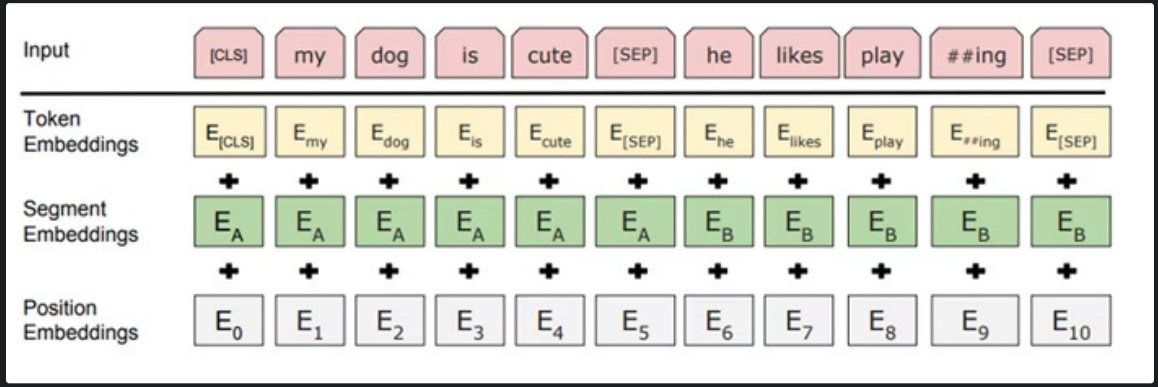
SST-2 sentiment classification

BERT Architecture




```python
class HeartBERTConfig:
    def __init__(
        self,
        vocab_size: int,
        hidden_size: int = 256,
        num_hidden_layers: int = 4,
        num_attention_heads: int = 4,
        intermediate_size: int = 1024,
        max_position_embeddings: int = 128,
        hidden_dropout_prob: float = 0.1,
        attention_probs_dropout_prob: float = 0.1,
        type_vocab_size: int = 2,
        initializer_range: float = 0.02,
    ):
```

Tokenizer & Input Embedding

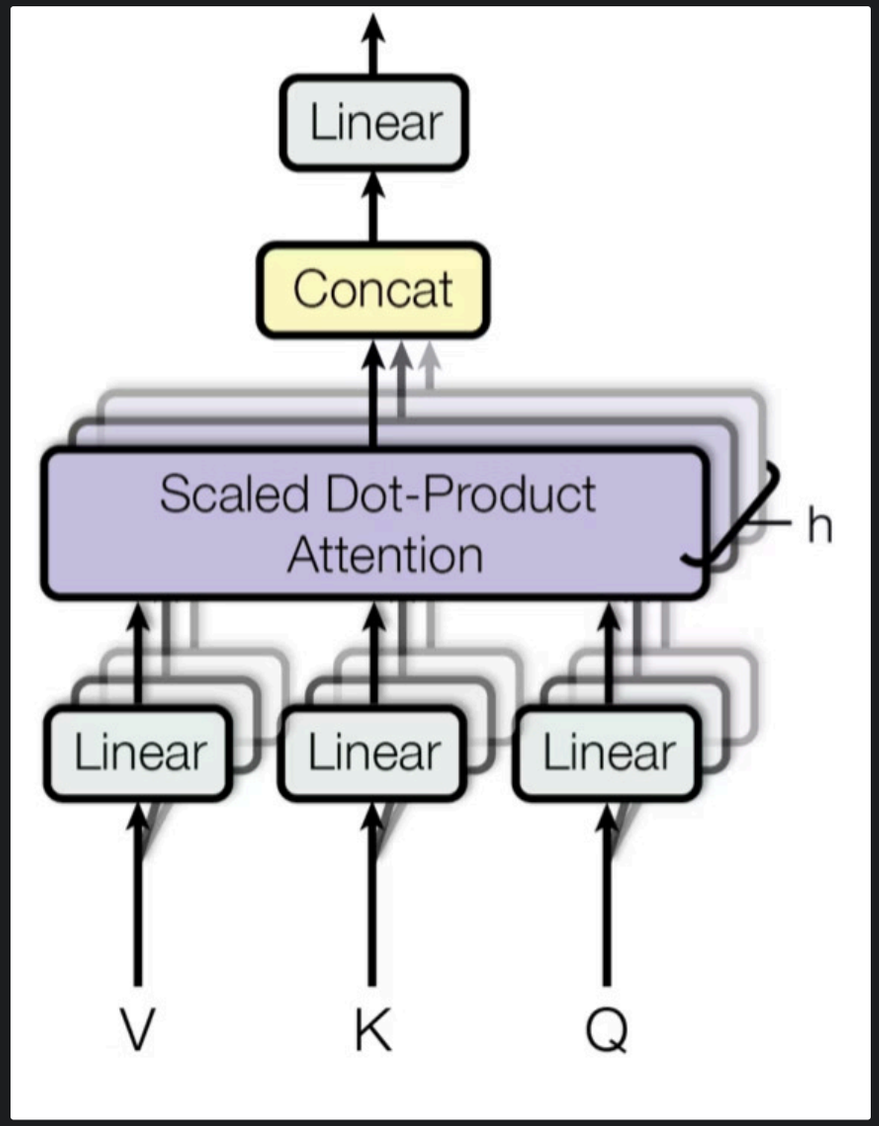



```python
class BERTModel(nn.Module):
    def __init__(self, config: HeartBERTConfig):
        super().__init__()
        self.config = config

        # Token Embeddings
        self.word_embeddings = nn.Embedding(config.vocab_size, config.hidden_size)
        # Position Embeddings
        self.position_embeddings = nn.Embedding(config.max_position_embeddings, config.hidden_size)
        # Token_type Embeddings (segment)
        self.token_type_embeddings = nn.Embedding(config.type_vocab_size, config.hidden_size)

        # LayerNorm & Dropout
        self.embeddings_norm = nn.LayerNorm(config.hidden_size, eps=1e-12)
        self.embeddings_dropout = nn.Dropout(config.hidden_dropout_prob)
```

Multi-head Self-Attention




```python
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, config: HeartBERTConfig):
        super().__init__()
        if config.hidden_size % config.num_attention_heads != 0:
            raise ValueError("hidden_size must be divisible by num_attention_heads")
        self.num_heads = config.num_attention_heads
        self.head_dim = config.hidden_size // config.num_attention_heads

        self.qkv = nn.Linear(config.hidden_size, config.hidden_size * 3)
        self.dropout = nn.Dropout(config.attention_probs_dropout_prob)
        self.out_proj = nn.Linear(config.hidden_size, config.hidden_size)

    def forward(self, hidden_states, attention_mask=None):
        batch_size, seq_len, _ = hidden_states.size()

        qkv = self.qkv(hidden_states)  # (batch, seq_len, 3 * hidden)
        query_layer, key_layer, value_layer = qkv.chunk(3, dim=-1)

        # (batch, seq_len, num_heads, head_dim) -> (batch, num_heads, seq_len, head_dim)
        def reshape_to_heads(x):
            return x.view(batch_size, seq_len, self.num_heads, self.head_dim).transpose(1, 2)

        query_layer = reshape_to_heads(query_layer)
        key_layer = reshape_to_heads(key_layer)
        value_layer = reshape_to_heads(value_layer)

        attention_scores = torch.matmul(query_layer, key_layer.transpose(-1, -2))
        attention_scores = attention_scores / math.sqrt(self.head_dim)

        if attention_mask is not None:
            attention_scores = attention_scores + attention_mask

        attention_probs = nn.Softmax(dim=-1)(attention_scores)
        attention_probs = self.dropout(attention_probs)

        context_layer = torch.matmul(attention_probs, value_layer)
        context_layer = context_layer.transpose(1, 2).contiguous()
        context_layer = context_layer.view(batch_size, seq_len, self.num_heads * self.head_dim)

        out = self.out_proj(context_layer)
        return out
```

Add & Norm + Feed-Forward Layer (FFN)

```
class TransformerBlock(nn.Module):

    def forward(self, hidden_states, attention_mask=None):
        attn_output = self.attention(hidden_states, attention_mask)
        attn_output = self.attention_dropout(attn_output)
        attn_output = self.attention_norm(hidden_states + attn_output)

        intermediate_output = self.intermediate(attn_output)
        intermediate_output = self.intermediate_act_fn(intermediate_output)

        layer_output = self.output_dense(intermediate_output)
        layer_output = self.output_dropout(layer_output)
        layer_output = self.output_norm(attn_output + layer_output)
        return layer_output
```

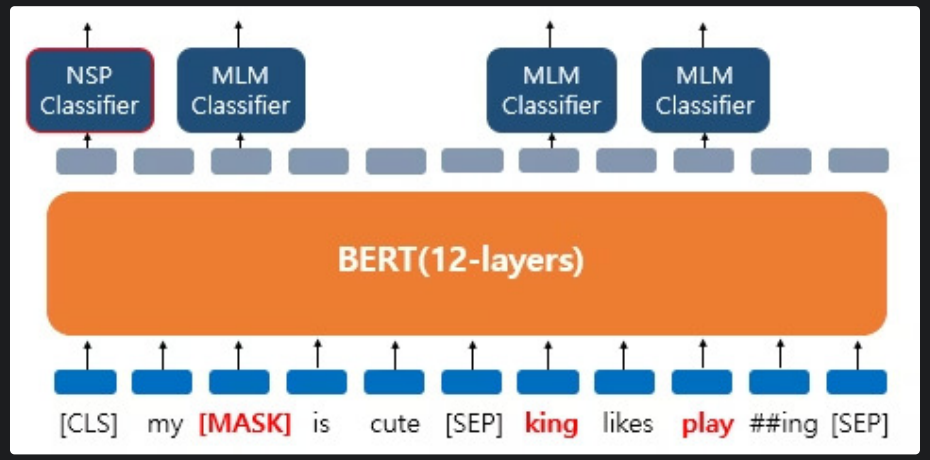
Pretraining – MLM

```python
class BERTForPreTraining(nn.Module):

    def forward(self, input_ids, token_type_ids=None, attention_mask=None, mlm_labels=None):
        sequence_output = self.bert(input_ids, token_type_ids, attention_mask)  # (batch, seq_len, h
        prediction_scores = self.mlm_dense(sequence_output)
        prediction_scores = self.activation(prediction_scores)
        prediction_scores = self.layer_norm(prediction_scores)
        prediction_scores = self.mlm_classifier(prediction_scores)  # (batch, seq_len, vocab_size)

        loss = None
        if mlm_labels is not None:
            # mlm_labels: (batch, seq_len), -100이 아닌 위치만 로스 계산
            loss_fct = nn.CrossEntropyLoss(ignore_index=-100)
            # (batch * seq_len, vocab_size) vs (batch * seq_len,)
            loss = loss_fct(prediction_scores.view(-1, self.config.vocab_size), mlm_labels.view(-1))

        return prediction_scores, loss
```



dataset:

- Hugging Face Wikipedia corpus (20220301.en)
- Sample of 100,000 sentences

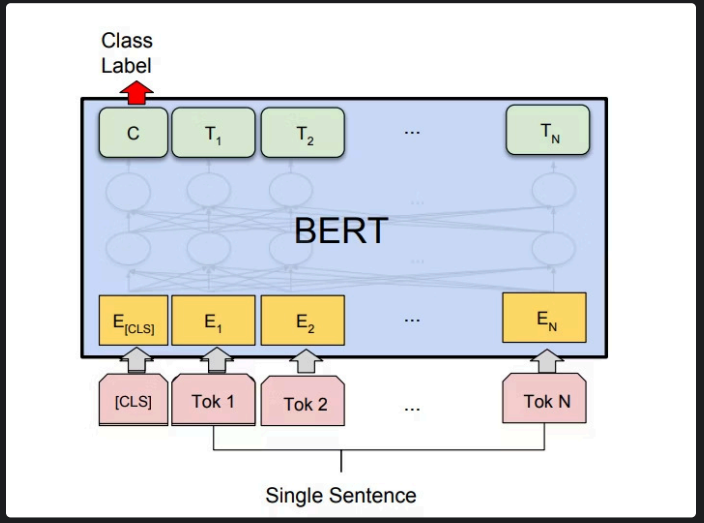
Fine Tuning - Classification Task

```python
class BERTForSequenceClassification(nn.Module):

    def forward(self, input_ids, token_type_ids=None, attention_mask=None, labels=None):
        sequence_output = self.bert(input_ids, token_type_ids, attention_mask)  # (batch,
        cls_output = sequence_output[:, 0, :]  # (batch, hidden)
        cls_output = self.dropout(cls_output)
        logits = self.classifier(cls_output)  # (batch, num_labels)

        loss = None
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, logits.size(-1)), labels.view(-1))

        return logits, loss
```

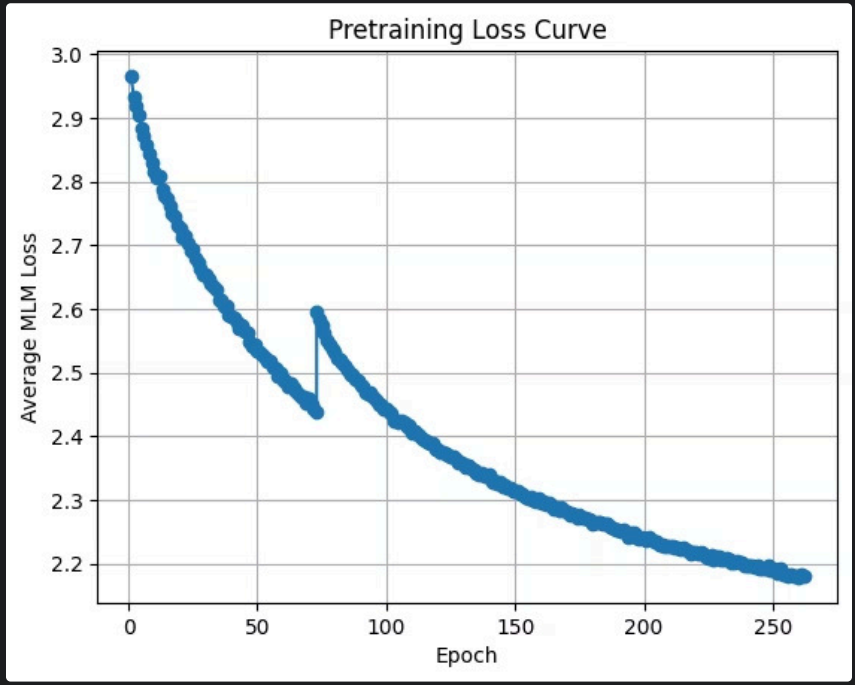


dataset:

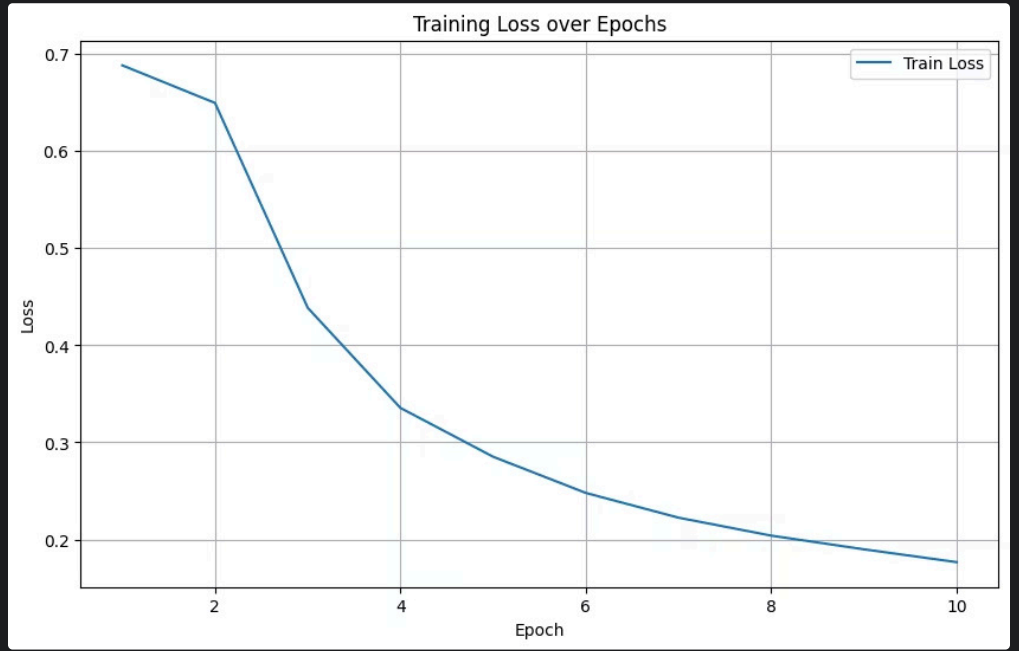
- GLUE benchmark, **SST-2** (Stanford Sentiment Treebank v2)
- binary sentiment classification (positive / negative)

Results - Training

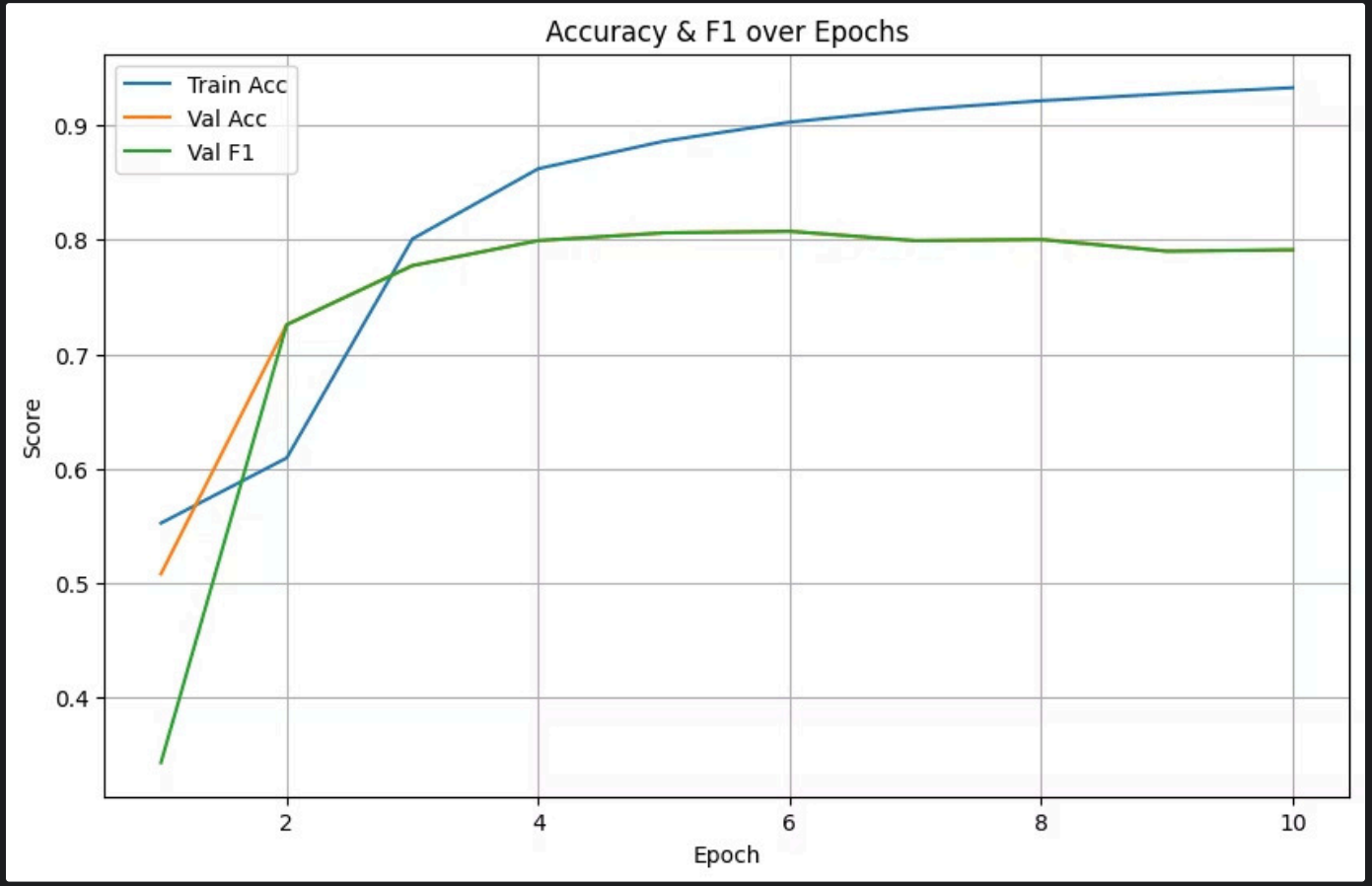
Pre-Training



Fine-Tuning



Results – Validation



Conclusion & Challenges

Implementing BERT from Scratch

Built the full BERT architecture manually, including embedding, attention, and transformer blocks.

End-to-End Pretraining and Finetuning

Completed the full pipeline from MLM-based pretraining to task-specific finetuning on SST-2

Efficient Dataset Loading with Chunking

Handled memory limitations by loading large datasets with memory-map method.