FLOPs and MACs Calculation for DistilBERT-Base

We will calculate the floating-point operations (FLOPs) and multiply-accumulate operations (MACs) for a single forward pass of the DistilBERT-Base model (6 layers, 12 attention heads, hidden size 768) on the following input sentence:

"I love deep learning. It's very interesting."

We assume:

- WordPiece tokenization
- FP16 precision, where each multiply–add pair is considered a single MAC (multiply–accumulate).
- A hidden size $H = 768$
- 12 attention heads (heads = 12), so each head has dimensionality $d_k = H/\text{heads} = 64$
- An FFN inner dimension $I = 3072$ (i.e. 4×768)
- 6 Transformer layers (DistilBERT typically has half the layers of BERT).

1. Tokenization

Tokenize the sentence "I love deep learning. It's very interesting." using WordPiece, adding `[CLS]` and `[SEP]`:

$$[CLS],\ i,\ love,\ deep,\ learning,\ .,\ it,\ \text{``}\#\#'s\text{''},\ very,\ interesting,\ .,\ [SEP]$$

We obtain $N = 12$ tokens.

2. Embeddings

DistilBERT learns a token embedding and a positional embedding of size 768. We add these two vectors elementwise for each token:

$$\text{Embeddings} = \underbrace{\text{Lookup(token\_id)} + \text{Lookup(position)}}_{\text{768-dim each}}.$$

- Lookups are not counted as FLOPs, as they are essentially indexing.
- Adding two 768-dim vectors for each token costs 768 additions per token.
- For N tokens, that is $N \times 768$ FLOPs.

Hence, for $N = 12$:

$$\text{Embedding\_FLOPs} = 12 \times 768 = 9216 \quad \text{(0 MACs)}.$$

3. Multi-Head Self-Attention (per layer)

Each layer has a multi-head self-attention (MHA) mechanism with 12 heads. The hidden size is 768, and each head operates on vectors of length $d_k = 64$. We break it down:

Q, K, V projections. For each token (length 768), we compute Query (Q), Key (K), and Value (V) vectors by multiplying by weight matrices of shape 768×768 (one for Q, one for K, one for V):

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

Each is a matrix multiply for dimension $(N \times 768) \times (768 \times 768)$. For one projection, multiply and add counts are each about $N \times 768 \times 768$. For three projections (Q,K,V):

$$\text{Mult}_{QKV} = 3 \times N \times 768 \times 768, \quad \text{Add}_{QKV} \approx 3 \times N \times 768 \times (768 - 1).$$

The MAC count equals the number of multiplications. For $N = 12$:

$$\text{MACs}_{QKV} = 3 \times 12 \times 768 \times 768 = 21{,}233{,}664.$$

Total FLOPs is multiplications plus additions, i.e. $\approx$ 42,439,680.

Attention scores $Q \cdot K^\top$. We form AttnScores $= QK^\top$ for each head. Each head has $N \times 64$ Q and $N \times 64$ K, so $N \times N$ dot products of length 64 per head. Each dot product has 64 multiplies and 63 adds:

$$\text{Mult}_{QK} = \text{heads} \times N \times N \times 64, \quad \text{Add}_{QK} \approx \text{heads} \times N \times N \times 63.$$

Hence MACs equals the multiply count. For $N = 12$, 12 heads:

$$\text{MACs}_{QK} = 12 \times 12^2 \times 64 = 110{,}592.$$

Total FLOPs $\approx$ 219,456.

Scaling, Softmax. We scale by $1/\sqrt{64}$, then apply softmax along each row. Softmax includes exponentiations, sums, and divisions. The total here is on the order of a few thousand FLOPs, negligible compared to the large matrix multiplies.

Applying softmax$(QK^\top)$ to V. We multiply an $N \times N$ attention matrix by $N \times d_k$ to produce $N \times d_k$. Per head, that costs $N \times N \times d_k$ multiplies plus $(N \times N \times (d_k - 1))$ adds. For 12 heads, that is the same scale as $QK^\top$:

$$\text{MACs}_{Attn \times V} = \text{heads} \times N \times N \times 64 = 110{,}592.$$

Total FLOPs $\approx$ 219,456 again.

Output projection. The 12 heads are concatenated into $N \times 768$, then multiplied by a 768×768 output weight. This is again $N \times 768 \times 768$ multiplies and similar adds. For $N = 12$:

$$\text{MACs}_O = 12 \times 768 \times 768 = 7{,}077{,}888,$$

FLOPs $\approx$ 14,146,560.

Summarizing MHA per layer:

$$\text{MHA\_FLOPs} \approx 56.99\text{M},$$
$$\text{MHA\_MACs} \approx 28.53\text{M}.$$

4. Feed-Forward Network (per layer)

Each layer has a two-layer FFN with dimensions $768 \to 3072 \to 768$ and a GELU nonlinearity in between.

First linear: 768×3072. For each token, we multiply a 768-dim vector by a 768×3072 matrix, so 768×3072 multiplies per token plus additions. For N tokens,

$$\text{MACs}_{\text{ffn1}} = N \times 768 \times 3072.$$

For $N = 12$, that is 28,311,552 MACs. FLOPs is about twice that (mult + add) $\approx$ 56.6M.

GELU activation. GELU is more complex than ReLU. Approximate 4 FLOPs per element. For $N \times 3072$ elements, $12 \times 3072 \times 4 = 147{,}456$ FLOPs, negligible compared to the matrix multiplies.

Second linear: 3072×768. Similarly, $N \times 3072 \times 768$ multiplies and similar adds. Another 28,311,552 MACs. $\approx$ 56.6M FLOPs.

Hence total FFN per layer:

$$\text{FFN\_FLOPs} \approx 113.2\text{M}, \quad \text{FFN\_MACs} \approx 56.62\text{M}.$$

5. Layer Normalization (per layer)

We apply LayerNorm twice per layer (after MHA, after FFN). For each token (768-dim):

1. Compute mean (768 adds, 1 div).
2. Compute variance (768 sub, 768 mul, sum, 1 div).
3. Normalize each value, multiply by gamma, add beta (768 sub, 768 mul, 768 mul, 768 add, 1 sqrt, 1 div).

Total ≈ 6147 FLOPs per token per LayerNorm. For $N = 12$:

$$\text{FLOPs}_{\text{LayerNorm}} \approx 6147 \times 12 = 73{,}764 \quad \text{(per LN)}.$$

Two LN calls per layer: 147,528 FLOPs per layer. Negligible MACs.

6. Residual Connections (per layer)

Each sub-layer output is added to the original input (dimension 768) per token. Two residual connections (after MHA, after FFN):

$$\text{FLOPs}_{\text{residual}} = 2 \times N \times 768.$$

For $N = 12$, $2 \times 12 \times 768 = 18{,}432$ FLOPs per layer.

7. Total (per layer) and for 6 Layers

Combining each component for a single layer:

$$\begin{aligned}
\text{MHA_FLOPs} &\approx 56.99\text{M},\\
\text{FFN_FLOPs} &\approx 113.2\text{M},\\
\text{LayerNorm} &\approx 0.147\text{M},\\
\text{Residual} &\approx 0.018\text{M},\\
\Rightarrow \text{Per-layer FLOPs} &\approx 170.54\text{M}.
\end{aligned}$$

$$\begin{aligned}
\text{MHA_MACs} &\approx 28.53\text{M},\\
\text{FFN_MACs} &\approx 56.62\text{M},\\
\text{LayerNorm} &\approx 0,\\
\text{Residual} &\approx 0,\\
\Rightarrow \text{Per-layer MACs} &\approx 85.15\text{M}.
\end{aligned}$$

DistilBERT has 6 layers, so multiply by 6:

$$\begin{aligned}
\text{6-layer FLOPs} &\approx 6 \times 170.54\text{M} \approx 1.02324 \times 10^9,\\
\text{6-layer MACs} &\approx 6 \times 85.15\text{M} \approx 5.109 \times 10^8.
\end{aligned}$$

Finally, add the Embedding cost (9216 FLOPs), negligible. No MACs in embedding. So total is:

$$\boxed{1.023 \times 10^9 \text{ FLOPs} \quad \text{and} \quad 5.11 \times 10^8 \text{ MACs.}}$$

8. Python Code for Verification

```python
N = 12 # sequence length
H = 768 # hidden size
heads = 12
dk = H // heads # 64
I = 3072 # intermediate size (4*768)
L = 6 # number of layers

# Embedding: token + position (just 768 adds per token)
```

```
embedding_flops = N * H
embedding_macs = 0

# Q, K, V
QKV_mult = 3 * N * H * H
QKV_add = 3 * N * H * (H - 1)
QKV_flops = QKV_mult + QKV_add
QKV_macs = QKV_mult

# Attention scores (Q*K^T)
attn_scores_mult = heads * N * N * dk
attn_scores_add = heads * N * N * (dk - 1)
attn_scores_flops = attn_scores_mult + attn_scores_add
attn_scores_macs = attn_scores_mult

# Softmax (approx)
softmax_exp = heads * N * N
softmax_add = heads * N * (N - 1)
softmax_div = heads * N * N
softmax_flops = softmax_exp + softmax_add + softmax_div
softmax_macs = 0

# Attn * V
attnV_mult = heads * N * N * dk
attnV_add = heads * N * N * (dk - 1)
attnV_flops = attnV_mult + attnV_add
attnV_macs = attnV_mult

# Output projection (768->768)
out_proj_mult = N * H * H
out_proj_add = N * H * (H - 1)
out_proj_flops = out_proj_mult + out_proj_add
out_proj_macs = out_proj_mult

# MHA block
MHA_flops = QKV_flops + attn_scores_flops + softmax_flops + attnV_flops + out_proj_flops
MHA_macs = QKV_macs + attn_scores_macs + softmax_macs + attnV_macs + out_proj_macs

# FFN
# First linear (768->3072)
ffn1_mult = N * H * I
ffn1_add = N * I * (H - 1)
ffn1_flops = ffn1_mult + ffn1_add
ffn1_macs = ffn1_mult

# GELU approx (4 ops per element)
gelu_flops = 4 * N * I
gelu_macs = 0

# Second linear (3072->768)
ffn2_mult = N * I * H
ffn2_add = N * H * (I - 1)
ffn2_flops = ffn2_mult + ffn2_add
ffn2_macs = ffn2_mult

FFN_flops = ffn1_flops + gelu_flops + ffn2_flops
FFN_macs = ffn1_macs + ffn2_macs

# LayerNorm (2x per layer): ~6147 FLOPs per token
ln_flops_per_token = 6147
ln_flops_per_layer = 2 * N * ln_flops_per_token
```

```
ln_macs_per_layer = 0

# Residual additions (2x per layer)
res_flops_per_layer = 2 * N * H
res_macs_per_layer = 0

# Combine per layer
flops_per_layer = MHA_flops + FFN_flops + ln_flops_per_layer + res_flops_per_layer
macs_per_layer = MHA_macs + FFN_macs + ln_macs_per_layer + res_macs_per_layer

# For 6 layers + embedding
total_flops = L * flops_per_layer + embedding_flops
total_macs = L * macs_per_layer + embedding_macs

print(f\"FLOPs per layer: {flops_per_layer}\")
print(f\"MACs per layer: {macs_per_layer}\")
print(f\"Total FLOPs for 6 layers + embedding: {total_flops}\")
print(f\"Total MACs for 6 layers + embedding: {total_macs}\")
```

Running this yields:

```
FLOPs per layer: 170543736
MACs  per layer: 85155840
Total FLOPs for 6 layers + embedding: 1023271632
Total MACs  for 6 layers + embedding: 510935040
```

Which matches our approximate arithmetic:

$$1.023 \times 10^9 \text{ FLOPs}, \quad 5.109 \times 10^8 \text{ MACs}.$$