I. Desain Arsitektur

Implementasi Transformer decoder-only ini dibangun sepenuhnya dari nol menggunakan NumPy tanpa bergantung pada library deep learning seperti PyTorch atau TensorFlow. Arsitektur yang dikembangkan mengikuti pendekatan GPT-style yang dirancang untuk tugas text generation dan language modeling. Model ini terdiri dari serangkaian komponen fundamental yang terintegrasi dengan baik, dimulai dari

a. proses token embedding yang mengonversi token indices menjadi dense vector representations
b. Penambahan positional encoding untuk memberikan informasi mengenai urutan token dalam sequence.
c. Data melewati multiple decoder layers yang masing-masing mengandung mekanisme multi-head attention dan feed-forward network dengan residual connections dan layer normalization.
d. Proses diakhiri dengan output projection ke vocabulary size dan normalisasi softmax untuk menghasilkan distribusi probabilitas yang dapat digunakan untuk memprediksi token berikutnya.

Desain ini memastikan bahwa model dapat menangkap dependensi jangka panjang dalam sequence sambil mempertahankan efisiensi komputasi.

II. Positional Encoding

Pemilihan sinusoidal positional encoding didasarkan pada pertimbangan mendalam mengenai kebutuhan model dalam menangani informasi posisional. Berbeda dengan learned positional encoding yang memiliki keterbatasan dalam menangani sequence length yang lebih panjang dari yang dilihat selama training, sinusoidal encoding memberikan kemampuan extrapolasi yang lebih baik. Pendekatan ini bersifat deterministik tanpa menambah parameter yang perlu dipelajari sehingga lebih efisien dalam penggunaan memory dan komputasi.

```
    for pos in range(seq_len):
        for i in range(0, embedding_dim, 2):
            pos_enc[pos, i] = np.sin(pos / (10000 ** (i / embedding_dim)))
            if i + 1 < embedding_dim:
                pos_enc[pos, i + 1] = np.cos(pos / (10000 ** (i / embedding_dim)))

    return pos_enc
```

Implementasinya menggunakan pola matematis dimana dimensi genap menggunakan fungsi sinus dan dimensi ganjil menggunakan fungsi kosinus, dengan frekuensi yang menurun secara eksponensial seiring meningkatnya dimensi embedding. Pola ini menciptakan representasi posisi yang unik untuk setiap posisi dalam sequence sekaligus memungkinkan model untuk mempelajari hubungan relatif antara posisi yang berbeda melalui operasi linear transformation.

III. Causal Masking

Causal masking merupakan komponen untuk mempertahankan sifat *autoregressive* dalam model transformer. Mekanisme ini memastikan bahwa selama proses attention, setiap token hanya dapat mengakses informasi dari token sebelumnya dan token saat ini tanpa dapat "melihat" token yang akan datang di masa depan.

```
def causal_mask(seq_len):

    return np.tril(np.ones((seq_len, seq_len)))
```

Implementasi teknik ini menggunakan lower triangular matrix dimana nilai 1 menunjukkan posisi yang diizinkan untuk diakses dan nilai 0 untuk posisi yang diblokir. Dalam mekanisme attention, posisi yang diblokir di-set ke nilai negatif yang sangat besar sebelum melalui fungsi softmax sehingga probabilitasnya menjadi mendekati nol setelah proses normalisasi.

```
=== Testing Multi-Head Attention ===
Input embeddings shape: (4, 8)
Output shape: (4, 8)
Number of attention heads: 2
Multi-Head Attention test passed!

=== Testing Causal Mask ===
Causal mask for sequence length 4
[[1. 0. 0. 0.]
 [1. 1. 0. 0.]
 [1. 1. 1. 0.]
 [1. 1. 1. 1.]]
Causal Mask test passed!
```

Pendekatan ini mensimulasikan kondisi nyata dimana model harus memprediksi token berikutnya hanya berdasarkan konteks yang telah dilihat sebelumnya, sekaligus mencegah terjadinya data leakage selama proses training yang dapat mengakibatkan overfitting.

IV. Bukti Uji Sederhana

Hasil pengujian menunjukkan bahwa seluruh komponen transformer berfungsi dengan baik dan sesuai dengan spesifikasi yang ditetapkan. Verifikasi dimensi tensor menunjukan konsistensi shape dari input hingga output, dimana token embedding berhasil mengubah input dengan dimensi [batch_size, seq_len] menjadi [batch_size, seq_len, embedding_dim], dan melalui serangkaian decoder layers dimensi tersebut dipertahankan hingga akhirnya diproyeksikan ke [batch_size, seq_len, vocab_size].

```
=== VERIFIKASI DIMENSI TENSOR ===
Input shape: (2, 5)
Logits shape: (2, 5, 1000)
Probabilities shape: (2, 5, 1000)
Number of attention layers: 2
 SEMUA DIMENSI SESUAI EKSPEKTASI
```

Pengujian fungsi softmax mengkonfirmasi bahwa proses normalisasi bekerja dengan stabil secara numerik dan menghasilkan distribusi probabilitas yang valid dimana jumlah probabilitas per posisi tepat 1.0.

```
=== VERIFIKASI SOFTMAX ===
Input scores:
[[1.  2.  3. ]
 [0.5 1.5 2.5]]
Output probabilities:
[[0.09003057 0.24472847 0.66524096]
 [0.09003057 0.24472847 0.66524096]]
Sum per row: [1. 1.]
Large scores sum: 1.0
 SOFTMAX BERFUNGSI DENGAN STABIL
```

Verifikasi causal masking membuktikan bahwa mekanisme autoregressive terjaga dengan sempurna, dimana attention weights untuk posisi masa depan selalu nol. Layer normalization juga berhasil menormalkan distribusi output dengan mean mendekati nol dan standard deviation mendekati satu, menunjukkan stabilitas numerik yang baik. Pengujian end-to-end menegaskan bahwa seluruh pipeline bekerja secara integratif dan konsisten, mulai dari pemrosesan input, transformasi melalui multiple decoder layers, hingga menghasilkan output probabilities yang dapat diandalkan untuk tugas text generation.4

```
=== VERIFIKASI CAUSAL MASKING ===
Causal Mask (seq_len=4):
[[1. 0. 0. 0.]
 [1. 1. 0. 0.]
 [1. 1. 1. 0.]
 [1. 1. 1. 1.]]
Attention weights dengan causal mask:
[[1.         0.         0.         0.        ]
 [0.03551605 0.96448395 0.         0.        ]
 [0.51965515 0.1332089  0.34713595 0.        ]
 [0.24728609 0.25089711 0.20483414 0.29698266]]
Upper triangular zero: True
CAUSAL MASKING BERFUNGSI DENGAN BAIK
```