Laporan Implementasi Decoder-Only Transformer dari Nol Menggunakan NumPy

Tugas Mata Kuliah Pemrosesan Bahasa Alami
Barbara Neanake Ajiesti (22/494495/TK/54238)
Teknologi Informasi, Universitas Gadjah Mada

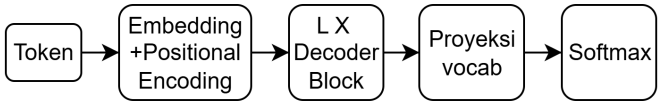
1 Pendahuluan

Sebagai asisten dosen, saya sering menerima pertanyaan serupa dari mahasiswa menjelang ujian. Dari pengalaman itu lahir ide *Smart AsDos*, prototipe sederhana yang dapat mengklasifikasikan pertanyaan umum mahasiswa untuk mengotomatiskan respon. Proyek ini saya gunakan sebagai konteks penerapan tugas *Transformer*, dengan fokus utama membangun dan memahami arsitektur *decoder-only Transformer* (GPT-style) sepenuhnya dari nol menggunakan NumPy.

Seluruh komponen, mulai dari *token embedding*, *sinusoidal positional encoding*, *multi-head self-attention*, *feed-forward network*, hingga *residual connection* dan *layer normalization*, saya rancang secara modular tanpa bantuan *deep learning library*. Implementasi ini tidak hanya memenuhi *requirement* tugas, tetapi juga membantu saya memahami kembali bagaimana model generatif modern bekerja dari dasar, dari embedding hingga pembentukan distribusi probabilitas token berikutnya.

2 Desain Arsitektur

Model yang saya bangun mengikuti alur:



Gambar 1: Alur asitektur decoder-only Transformer Barbara

Seluruh implementasi ditulis menggunakan *NumPy-only*, tanpa *libary* modern, sehingga seluruh operasi linear, seperti normalisasi dan masking, dilakukan manual. Pendekatan modular ini memudahkan proses debugging dan validasi dimensi tiap lapisan.

Tabel 1: Ringkasan komponen dan alasan desain arsitektur.

Komponen	. Implementasi	Alasan Pemilihan
Token Embedding	Matriks (*vocab_size*, *d_model*); lookup berdasarkan indeks token.	Lebih efisien dibanding one-hot dan memungkinkan *weight tying* dengan layer output untuk menjaga konsistensi representasi.
Positional Encoding	Menggunakan versi *sinusoidal* seperti pada paper asli Transformer.	Tidak menambah parameter baru dan dapat mengeneralisasi ke panjang sekuens lebih besar dari data pelatihan.
Decoder Block (Pre-Norm)	LayerNorm → MHA (+residual) → LayerNorm → FFN (+residual).	Pre-norm meningkatkan stabilitas numerik dan mencegah *gradient vanishing* saat model ditumpuk.
Multi-Head Self-Attention	Proyeksi Q, K, V → scaled dot-product → concat → proyeksi W_o.	Menangkap pola relasi antar token pada subruang representasi berbeda secara paralel, dengan dukungan *causal mask*.
Feed-Forward Network (FFN)	Dua layer linear: $d_{model} \to d_{ff} \to d_{model}$ dengan aktivasi ReLU.	Sederhana namun efektif untuk memperkaya representasi per posisi secara independen.
Residual + LayerNorm	Residual di setiap sublayer; normalisasi dilakukan per fitur.	Menjaga kestabilan distribusi nilai antar layer dan mempertahankan informasi kontekstual awal.
Causal Mask	Mask segitiga bawah (*lower-triangular*) diterapkan pada skor $QK^\top$.	Menjamin model bersifat *autoregressive*, di mana token hanya dapat memperhatikan token sebelumnya.
Output + Softmax	Proyeksi ke ukuran *vocab_size* lalu *softmax*; menggunakan *weight tying*.	Menghemat parameter dan menjaga konsistensi antara ruang embedding dan output.
(Bonus) Visualisasi Attention	*Heatmap* antar head divisualisasikan dengan `matplotlib`.	Membantu interpretasi perilaku perhatian serta memverifikasi efektivitas *causal masking*.

3 Alasan Pemilihan Positional Encoding

Dalam implementasi ini saya menggunakan *sinusoidal positional encoding*, sesuai penjelasan pada kuliah Transformer (Vaswani et al., 2017; Jurafsky & Martin, 2024). Pemilihan metode ini didasarkan pada tiga alas-

an: (1) tidak menambah parameter baru, (2) mampu mengeneralisasi ke panjang sekuens di luar data pelatihan, dan (3) menghasilkan pola periodik yang memudahkan model membedakan urutan posisi token.

Secara formal, setiap posisi *pos* dan dimensi i dihitung sebagai:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right),$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right).$$

Pada proyek ini, fungsi tersebut saya implementasikan dengan NumPy dan ditambahkan langsung ke matriks embedding. Pendekatan ini lebih efisien dibandingkan *learned positional encoding* yang meski fleksibel, menambah parameter dan kompleksitas pelatihan, kurang relevan untuk proyek *from scratch* berskala kecil seperti *Smart AsDos*.

4 Causal Mask

Agar model bersifat *autoregressive*, setiap token hanya boleh memperhatikan dirinya dan token sebelumnya. Tanpa mekanisme ini, perhatian dapat menjangkau token masa depan, menyebabkan kebocoran konteks. Kalkulasi perhatian dalam bentuk matriks (*Eq. 9.32, Jurafsky & Martin, 2024*) adalah:

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V,$$

dengan $M_{ij} = -\infty$ untuk $j > i$ dan $M_{ij} = 0$ untuk $j \leq i$. Nilai $-\infty$ memastikan bahwa setelah *softmax*, bobot untuk posisi masa depan bernilai nol.

Dalam notebook saya, fungsi `create_causal_mask()` menghasilkan matriks segitiga bawah (*lower-triangular*) yang kemudian diverifikasi menggunakan `validate_masking()`, memastikan tidak ada perhatian melewati batas waktu. Masking ini menjadi kunci agar model *decoder-only* mampu memprediksi token berikutnya secara berurutan layaknya model GPT-style.

5 Pengujian dan Validasi

Uji `forward_pass` menggunakan input acak (`batch=2`, `seq_len=15`) menghasilkan *logits* dan *probabilities* berdimensi konsisten $(2, 15, |\mathcal{V}|)$. Fungsi `validate_masking()` tidak mendeteksi pelanggaran (`violations=0`), dan setiap distribusi *softmax* memenuhi $\sum_v P_{i,t,v} \approx 1.0$. Visualisasi `plot_layer_head_heatmaps()` juga menunjukkan fokus perhatian ke token kiri, sesuai sifat *autoregressive*. Cuplikan hasil uji ditunjukkan pada Gambar 2.

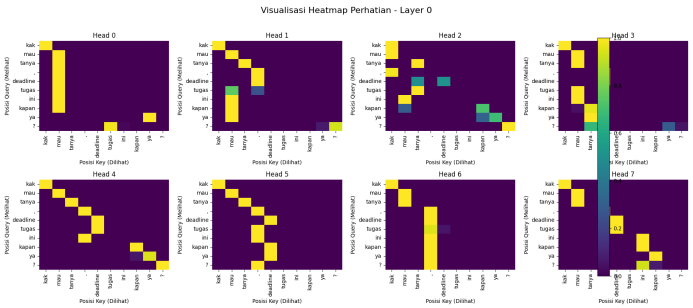
```
--- Bukti Uji 1: Verifikasi Dimensi Tensor ---
Bentuk input:
[[47  4 30 36 40 19 21 44 24 35 34 49 11 34 48]
 [34 49 27 21 16 38 34 18  6  5  4 22 41  4 22]]

Bentuk output logits: (2, 15, 50)  --> Sesuai (batch, seq_len, vocab_size)
Bentuk output probabilities: (2, 15, 50) --> Sesuai (batch, seq_len, vocab_size)
Jumlah layer perhatian yang disimpan: 4
Bentuk matriks perhatian (contoh Layer 0): (2, 8, 15, 15) --> Sesuai (batch, heads, seq, seq)
```

Gambar 2: Cuplikan hasil uji `forward_pass` dan verifikasi dimensi tensor.

Tabel 2: Ringkasan hasil pengujian model.

Jenis Uji	**Hasil Observasi**
Dimensi output	$(2, 15, \|\mathcal{V}\|)$
Softmax sum per token	1.000 ± 10^{-6}
Causal mask violation	0
Distribusi attention	Fokus ke token kiri



Gambar 3: Contoh visualisasi *attention head* yang dihasilkan model.

Tabel 3: Ringkasan hasil pengujian.

Jenis Uji	**Hasil**
Dimensi output	$(2, 15, \|\mathcal{V}\|)$
Softmax sum per token	≈ 1.000
Causal mask leak	0 (aman)
Attention fokus kiri	Terverifikasi via heatmap

6 Kesimpulan

Saya berhasil membangun model *decoder-only Transformer* sepenuhnya menggunakan NumPy, mulai dari embedding, positional encoding, multi-head attention, hingga causal masking. Seluruh komponen berfungsi sesuai rancangan dengan keluaran berdimensi konsisten $(2, 15, |\mathcal{V}|)$, distribusi *softmax* valid, dan tanpa pelanggaran masking.

Pendekatan *pre-norm* dan struktur modular membuat komputasi stabil meski dijalankan secara numerik murni. Proyek ini memperdalam pemahaman saya tentang prinsip GPT dan menunjukkan bagaimana arsitektur tersebut dapat disederhanakan untuk kasus nyata seperti *Smart AsDos*.

Repository: `https://github.com/barbaraneanake/Transformer-FromScratch`