Meeting 3: Komponen Transformer Block

AI-40X: Generative AI & Large Language Models

Hendri Karisma, M.T.
Dosen Teknik Informatika STMIK Tazkia
VP Engineering at jejakin.com, 2026

Semester 4 — 2025/2026

Outline

Mengapa Informasi Posisi Diperlukan?

- Berbeda dengan RNN yang memproses token **secara berurutan** (satu per satu), Transformer memproses **seluruh sequence sekaligus** secara paralel.
- Konsekuensinya: Transformer **tidak memiliki konsep urutan bawaan** (no recurrence, no convolution).
- Tanpa informasi posisi:
  - "Kucing mengejar tikus" $\equiv$ "Tikus mengejar kucing"
  - Self-Attention bersifat **permutation-equivariant**: mengubah urutan input hanya mengubah urutan output, bukan nilainya.
- **Solusi:** Tambahkan vektor *Positional Encoding* ke setiap token embedding:

$$\mathbf{z}_i = \mathbf{x}_i + \mathbf{PE}(\text{pos}_i)$$

dimana $\mathbf{x}_i \in \mathbb{R}^{d_{\text{model}}}$ adalah token embedding dan $\mathbf{PE}(\text{pos}_i) \in \mathbb{R}^{d_{\text{model}}}$ adalah positional encoding.

Sinusoidal Positional Encoding: Formula

Vaswani et al. (2017) mengusulkan encoding berbasis gelombang sinus dan cosinus:

Formula Positional Encoding

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

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

- pos: posisi token dalam sequence $(0, 1, 2, \ldots, L-1)$
- i: indeks dimensi $(0, 1, 2, \ldots, d_{\text{model}}/2 - 1)$
- Dimensi genap $(2i)$ menggunakan sin, dimensi ganjil $(2i+1)$ menggunakan cos
- Setiap dimensi memiliki **frekuensi berbeda**: dari wavelength 2π (dimensi awal) hingga $2\pi \cdot 10000$ (dimensi akhir)

Properti Sinusoidal Encoding

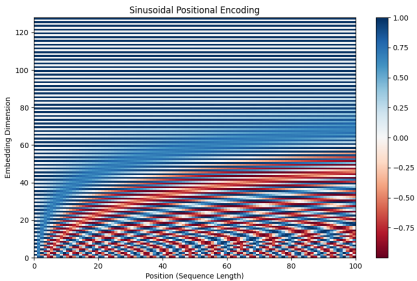
Keunggulan desain ini:

1. **Unik per posisi:** Setiap posisi mendapat vektor encoding yang berbeda (kombinasi frekuensi unik).
2. **Nilai terbatas (bounded):** Karena sin dan cos menghasilkan nilai $\in [-1, 1]$, encoding tidak akan "meledak" untuk posisi besar.
3. **Menangkap posisi relatif:** Untuk offset tetap k, $PE(\text{pos} + k)$ dapat dinyatakan sebagai **transformasi linear** dari $PE(\text{pos})$:

$$PE(\text{pos} + k) = T_k \cdot PE(\text{pos})$$

Model dapat belajar menangkap jarak relatif antar token.

4. **Generalisasi ke sequence lebih panjang** dari



Visualisasi PE: sumbu-x = dimensi, sumbu-y = posisi. Pola gelombang berubah frekuensi di setiap dimensi.

Learnable vs Sinusoidal Positional Encoding

Sinusoidal (Fixed)

- Tidak ada parameter tambahan
- Bisa generalisasi ke panjang sequence baru (extrapolation)
- Digunakan di Transformer asli (Vaswani et al., 2017)
- Deterministik — tidak tergantung data training

Learnable

- Menambah parameter: matriks $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{L_{\max} \times d_{\text{model}}}$
- Lebih fleksibel — model bebas belajar pola posisi optimal
- Digunakan oleh GPT, BERT
- **Kelemahan:** terbatas pada $L_{\max}$ yang ditentukan saat training

Temuan Empiris

Vaswani et al. melaporkan kedua metode menghasilkan performa yang **hampir identik**. Namun di era LLM modern, teknik baru seperti **RoPE** (Rotary Position Embedding) dan **ALiBi** menjadi standar karena kemampuan ekstrapolasi yang lebih baik.

Position-wise Feed-Forward Network

Setelah Multi-Head Attention mengumpulkan konteks antar token, informasi tersebut diproses oleh **FFN secara independen per posisi**:

Formula FFN

$$\text{FFN}(\mathbf{x}) = \max(0,\ \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\,\mathbf{W}_2 + \mathbf{b}_2$$

- $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{ff}}$ — proyeksi ke dimensi lebih tinggi
- $\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d_{\text{model}}}$ — proyeksi kembali ke dimensi model
- $\max(0, \cdot)$ adalah fungsi aktivasi ReLU
- **Position-wise:** FFN yang *sama* diterapkan pada setiap posisi token secara independen (seperti konvolusi 1×1)

Mengapa Ekspansi ke 4× Dimensi?

Pada Transformer asli: $d_{\text{model}} = 512$, $d_{ff} = 2048$ (rasio 4×).

- **Bottleneck architecture:** Expand → Activate → Compress

$$\underbrace{\mathbb{R}^{512}}_{\text{input}} \xrightarrow{\mathbf{W}_1} \underbrace{\mathbb{R}^{2048}}_{\text{hidden}} \xrightarrow{\text{ReLU}} \underbrace{\mathbb{R}^{2048}}_{\text{sparse}} \xrightarrow{\mathbf{W}_2} \underbrace{\mathbb{R}^{512}}_{\text{output}}$$

- **Proyeksi ke ruang dimensi tinggi** memungkinkan model merepresentasikan **kombinasi fitur** yang lebih kaya dan kompleks
- Fungsi aktivasi (ReLU) membuat representasi menjadi **sparse** — hanya sebagian neuron aktif, memungkinkan model menyimpan "pengetahuan" di neuron berbeda
- **Proyeksi kembali** ke d_{model} menjaga ukuran representasi tetap konsisten antar layer
- *Analogi:* Attention = "diskusi kelompok" (mengumpulkan info), FFN = "refleksi pribadi" (memproses info yang dikumpulkan)

Evolusi Fungsi Aktivasi: ReLU ke GELU

- **ReLU** (Transformer asli): $\text{ReLU}(x) = \max(0, x)$
  - Sederhana dan efisien, tapi bersifat "keras" (hard threshold di 0)
  - Neuron dengan input negatif sepenuhnya mati (dead neuron problem)
- **GELU** (GPT, BERT): $\text{GELU}(x) = x \cdot \Phi(x)$ dimana Φ adalah CDF distribusi normal standar
  - Transisi halus (smooth) di sekitar 0 — tidak ada kematian neuron mendadak
  - Secara stokastik "membuang" input kecil (mirip dropout implisit)
  - Menjadi standar de facto di hampir semua LLM modern
- **SwiGLU** (LLaMA, PaLM): $\text{SwiGLU}(\mathbf{x}) = (\mathbf{x}\mathbf{W}_1 \odot \text{Swish}(\mathbf{x}\mathbf{V}))\mathbf{W}_2$
  - Gated mechanism: menggunakan gate tambahan untuk mengontrol aliran informasi
  - Sedikit lebih mahal secara komputasi, tapi performa lebih baik

Fakta Penting

FFN menyimpan sekitar $\frac{2}{3}$ **dari total parameter** model Transformer! (2 matriks besar: $\mathbf{W}_1$ dan $\mathbf{W}_2$). FFN sering diinterpretasikan sebagai "memory" model — menyimpan pengetahuan faktual.

Residual Connection (Skip Connection)

- **Masalah deep network:** Semakin dalam jaringan, gradien bisa *vanish* (mengecil mendekati 0) saat backpropagation — layer awal tidak belajar.
- **Insight dari ResNet** (He et al., 2015): Tambahkan "jalan pintas" (shortcut) yang melewati sublayer:

$$\text{output} = x + \text{SubLayer}(x)$$

- **Mengapa ini membantu?** Gradien mendapat jalur langsung:

$$\frac{\partial \text{output}}{\partial x} = \frac{\partial(x + \text{SubLayer}(x))}{\partial x} = \underbrace{1}_{\text{shortcut}} + \underbrace{\frac{\partial \text{SubLayer}(x)}{\partial x}}_{\text{sublayer gradien}}$$

- Faktor "1" menjamin gradien **minimal bernilai 1** — tidak akan vanish total!
- Dalam Transformer, setiap sub-layer (MHA dan FFN) dibungkus residual connection.

Layer Normalization

Layer Normalization (Ba et al., 2016) menormalisasi aktivasi *di sepanjang dimensi fitur* untuk setiap token secara independen:

Formula Layer Normalization

$$\text{LayerNorm}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$$

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \qquad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2$$

- $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{d_{\text{model}}}$: parameter yang *learnable* (scale dan shift)
- ϵ: konstanta kecil untuk stabilitas numerik (biasanya 10^{-5})
- Normalisasi menjaga distribusi aktivasi tetap stabil antar layer — mempercepat dan menstabilkan training

Layer Norm vs Batch Norm

Batch Normalization

- Normalisasi *di sepanjang batch* untuk setiap fitur
- Bergantung pada ukuran batch — tidak stabil untuk batch kecil
- Memerlukan running statistics saat inference
- Populer di Computer Vision (CNN)

Layer Normalization

- Normalisasi *di sepanjang fitur* untuk setiap sample
- **Independen dari batch** — stabil di batch size berapa pun
- Tidak perlu running statistics
- Standar di Transformer/NLP karena panjang sequence bervariasi

Ringkasan Perbedaan

BatchNorm: normalisasi per-fitur lintas batch
LayerNorm: normalisasi per-sample lintas fitur

Pre-Norm vs Post-Norm

Dua varian penempatan LayerNorm dalam Transformer:

Post-Norm (Transformer asli):

$$\mathbf{y} = \text{LN}(\mathbf{x} + \text{SubLayer}(\mathbf{x}))$$

- LayerNorm *setelah* residual addition
- Memerlukan **learning rate warmup** yang hati-hati
- Training kurang stabil, tapi hasil akhir sering sedikit lebih baik

Pre-Norm (GPT-2, LLaMA):

$$\mathbf{y} = \mathbf{x} + \text{SubLayer}(\text{LN}(\mathbf{x}))$$

- LayerNorm *sebelum* sublayer
- Training **jauh lebih stabil** — tidak perlu warmup rumit
- Menjadi standar di hampir semua LLM modern

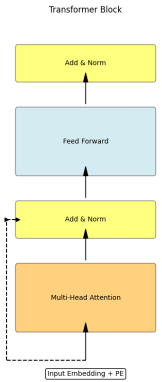
Catatan

Xiong et al. (2020) menunjukkan Pre-Norm mengatasi masalah "gradient explosion at initialization" yang dialami Post-Norm, sehingga memungkinkan training model yang sangat

Diagram Transformer Block (Pre-Norm)

Alur data dalam satu Transformer Block:

1. **Input x**
2. $\mathbf{x}' = \text{LayerNorm}(\mathbf{x})$
3. $\mathbf{a} = \text{MultiHeadAttn}(\mathbf{x}', \mathbf{x}', \mathbf{x}')$
4. $\mathbf{h} = \mathbf{x} + \mathbf{a}$ (Residual)
5. $\mathbf{h}' = \text{LayerNorm}(\mathbf{h})$
6. $\mathbf{f} = \text{FFN}(\mathbf{h}')$
7. $\mathbf{output} = \mathbf{h} + \mathbf{f}$ (Residual)



Ringkasan:

Input → LN → MHA → Add → LN → FFN → Add → Outp

Menumpuk N Transformer Block

- Satu Transformer Block sudah cukup ekspresif, tetapi **menumpuk N block** memungkinkan model membangun representasi yang semakin abstrak:
  - **Layer bawah:** menangkap pola lokal (sintaksis, n-gram)
  - **Layer tengah:** menangkap hubungan semantik (sinonim, koreferensi)
  - **Layer atas:** menangkap penalaran tingkat tinggi (logika, konteks global)
- **Contoh arsitektur:**
  - Transformer asli: $N = 6$ layers
  - GPT-2 Small: $N = 12$, GPT-2 Large: $N = 36$
  - GPT-3: $N = 96$ layers
  - LLaMA-2 70B: $N = 80$ layers
- Output dari block terakhir diumpankan ke **task-specific head** (misal: Linear + Softmax untuk language modeling)
- Residual connections memastikan gradien tetap mengalir lancar meskipun jaringan sangat dalam

Menghitung Jumlah Parameter

Untuk satu Transformer Block dengan $d_{\text{model}} = d$ dan $d_{ff} = 4d$:

Komponen	**Parameter**	**Contoh (d=768)**
MHA (W^Q, W^K, W^V, W^O)	$4d^2$	2.4M
FFN (W_1, W_2)	$8d^2$	4.7M
LayerNorm ($\times 2$)	$4d$	3K
Total per block	$\approx 12d^2$	$\approx$ 7.1M

- FFN menyumbang $\frac{8d^2}{12d^2} = \frac{2}{3}$ dari parameter per block!
- Untuk N block: total $\approx 12Nd^2$ parameter (belum termasuk embedding)
- **Contoh GPT-2 Small:** 12 block $\times$ 7.1M $\approx$ 85M + embedding $\approx$ **124M** parameter

Kesimpulan Pertemuan 3

1. **Positional Encoding** memberikan informasi urutan yang hilang akibat arsitektur paralel Transformer — bisa sinusoidal (fixed) atau learnable.
2. **Feed-Forward Network** memproses setiap posisi secara independen dengan ekspansi $4\times$ dimensi, menyimpan $\approx \frac{2}{3}$ parameter model. Evolusi aktivasi: ReLU $\rightarrow$ GELU $\rightarrow$ SwiGLU.
3. **Residual Connections** memastikan gradien mengalir lancar ke layer terdalam, memungkinkan arsitektur sangat dalam ($N > 90$ layer).
4. **Layer Normalization** menstabilkan distribusi aktivasi. Pre-Norm lebih stabil dari Post-Norm.
5. Satu **Transformer Block lengkap**: LN $\rightarrow$ MHA $\rightarrow$ Add $\rightarrow$ LN $\rightarrow$ FFN $\rightarrow$ Add, ditumpuk N kali.

Pertemuan berikutnya: Tokenization dan Arsitektur GPT (Decoder-Only Transformer).

Referensi

- Vaswani, A., et al. (2017). “Attention Is All You Need.” *NeurIPS*.
- He, K., et al. (2016). “Deep Residual Learning for Image Recognition.” *CVPR*.
- Ba, J.L., Kiros, J.R., & Hinton, G.E. (2016). “Layer Normalization.” *arXiv:1607.06450*.
- Xiong, R., et al. (2020). “On Layer Normalization in the Transformer Architecture.” *ICML*.
- Hendrycks, D. & Gimpel, K. (2016). “Gaussian Error Linear Units (GELUs).” *arXiv:1606.08415*.
- Shazeer, N. (2020). “GLU Variants Improve Transformer.” *arXiv:2002.05202*.
- Alammar, J. “The Illustrated Transformer.” `https://jalammar.github.io/illustrated-transformer/`