Meeting 4: Tokenization & Arsitektur GPT

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

Dari Teks ke Angka: Langkah Pertama

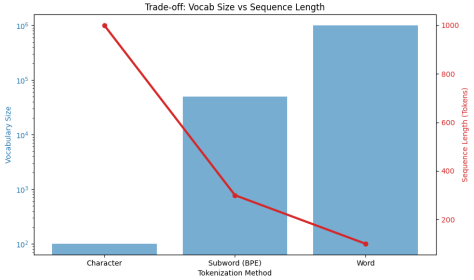
- Model neural network hanya bisa memproses **angka** (tensor), bukan teks mentah.
- **Tokenization:** proses mengubah string teks menjadi urutan *token ID* (integer) berdasarkan kamus (*vocabulary*).

$$\text{“Saya belajar AI”} \xrightarrow{\text{tokenizer}} [1547,\ 892,\ 34] \xrightarrow{\text{embedding}} \begin{bmatrix} \mathbf{e}_{1547} \\ \mathbf{e}_{892} \\ \mathbf{e}_{34} \end{bmatrix}$$

- Tokenization adalah **langkah paling pertama** dalam pipeline LLM — sebelum embedding, sebelum attention, sebelum segalanya.
- Kualitas tokenizer secara langsung mempengaruhi:
  - Efisiensi komputasi (panjang sequence)
  - Kemampuan menangani kata baru / bahasa asing
  - Biaya inference (dihitung per token!)

Trade-off: Ukuran Vocabulary vs Panjang Sequence

- **Vocabulary besar** (word-level):
  - Sequence pendek (efisien)
  - Tapi: embedding table raksasa, banyak kata jarang tidak tercover (OOV)
- **Vocabulary kecil** (character-level):
  - Tidak ada OOV — semua teks bisa direpresentasikan
  - Tapi: sequence sangat panjang, model harus belajar "mengeja"
- **Sweet spot:** Subword tokenization (∼30K–100K token)
  - Sequence moderat
  - Vocabulary moderat
  - Kata umum = 1 token, kata jarang = beberapa subword



Trade-off antara ukuran vocabulary dan panjang sequence rata-rata.

Character-Level Tokenization

- **Ide:** Setiap karakter (huruf, angka, simbol) menjadi satu token.
- Vocabulary sangat kecil: $\sim$256 (ASCII) atau $\sim$150K (full Unicode)
- **Kelebihan:**
  - Zero OOV — bisa merepresentasikan teks apa pun
  - Vocabulary table kecil (hemat memori embedding)
- **Kekurangan:**
  - Sequence menjadi **sangat panjang**: "Universitas" = 11 token!
  - Self-attention berskala $O(n^2)$ — sequence panjang sangat mahal
  - Model harus belajar menyusun makna dari level huruf (sangat sulit)
- **Contoh:**

$$\text{"kucing"} \rightarrow [\texttt{k}, \texttt{u}, \texttt{c}, \texttt{i}, \texttt{n}, \texttt{g}] \quad (6 \text{ token})$$

Word-Level Tokenization

- **Ide:** Setiap kata utuh menjadi satu token. Pemisahan biasanya berdasarkan spasi/tanda baca.
- **Kelebihan:**
  - Sequence pendek — setiap kata langsung punya makna
  - Intuitif dan mudah dipahami
- **Kekurangan kritis — Out-of-Vocabulary (OOV):**
  - Kata yang tidak ada di vocabulary → <UNK> (informasi hilang total!)
  - Bahasa Indonesia sangat produktif secara morfologi: "mempelajari", "pembelajaran", "dipelajari" — semua token berbeda
  - Nama, istilah teknis, typo → semuanya menjadi <UNK>
  - Vocabulary harus **sangat besar** (>500K) untuk cukup, tapi embedding table menjadi tidak praktis
- **Contoh:**

"kucing berlari cepat" → [`kucing`, `berlari`, `cepat`] (3 token)

Subword Tokenization: Solusi Terbaik

- **Ide kunci:** Kata umum disimpan utuh, kata jarang **dipecah** menjadi potongan bermakna (subword).
- Memberikan keseimbangan optimal antara vocabulary size dan sequence length.
- **Algoritma utama:**
  1. **BPE** (Byte Pair Encoding) — digunakan oleh GPT-2, GPT-3, GPT-4
  2. **WordPiece** — digunakan oleh BERT, DistilBERT
     - Mirip BPE, tapi merge berdasarkan *likelihood* (mutual information), bukan frekuensi mentah
     - Menandai potongan lanjutan dengan prefix **##**: "playing" → ["play", "##ing"]
  3. **SentencePiece** — digunakan oleh T5, LLaMA, mBERT
     - Bekerja langsung pada teks mentah (raw text) tanpa pre-tokenization
     - Language-agnostic: tidak bergantung pada spasi sebagai pemisah kata
     - Sangat cocok untuk bahasa non-Latin (Jepang, China, Thai)
- **Contoh BPE:**

"mempelajari" → [`mem`, `pel`, `ajari`] (3 subword)

BPE: Intuisi Dasar

Byte Pair Encoding (Sennrich et al., 2016) — diadaptasi dari algoritma kompresi data:

Ide Utama

Mulai dari karakter individual, lalu **berulang kali menggabungkan** pasangan simbol yang **paling sering muncul** bersamaan, sampai ukuran vocabulary yang diinginkan tercapai.

Langkah-langkah:

1. **Inisialisasi:** Pecah seluruh korpus menjadi karakter individual. Vocabulary awal = semua karakter unik.
2. **Hitung frekuensi:** Hitung frekuensi semua pasangan simbol bersebelahan (bigram) dalam korpus.
3. **Merge:** Gabungkan pasangan paling sering menjadi simbol baru. Tambahkan ke vocabulary.
4. **Ulangi:** Kembali ke langkah 2 dengan vocabulary yang diperbarui.
5. **Berhenti:** Ketika vocabulary mencapai ukuran target (misal 50.257 untuk GPT-2).

Contoh BPE dengan Teks Indonesia

Korpus: "belajar belajar bekerja bekerja bekerja"

Langkah 0 — Inisialisasi (pecah ke karakter):
`b e l a j a r _ b e l a j a r _ b e k e r j a _ b e k e r j a _ b e k e r j a`
Vocab: {b, e, l, a, j, r, k, _}

Langkah 1 — Pasangan tersering: (e, r) muncul 5×
Merge: `e r` → `er`
Vocab: {b, e, l, a, j, r, k, _, **er**}

Langkah 2 — Pasangan tersering: (er, j) muncul 3× atau (b, e) muncul 5×
Merge: `b e` → `be`
Vocab: {b, e, l, a, j, r, k, _, er, **be**}

Proses berlanjut... hingga terbentuk token seperti `belajar`, `bekerja` sebagai token utuh (karena sering muncul).

Visualisasi Merge Tree

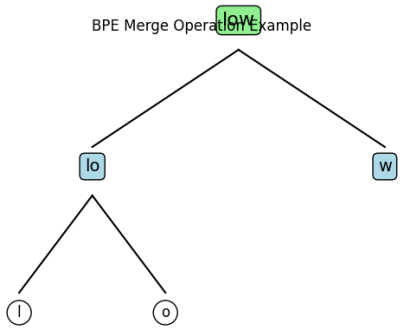
Proses merge membentuk pohon (tree):

- Daun (leaf) = karakter individual
- Setiap merge = node internal baru
- Akar (root) = token final

Contoh untuk "belajar":

1. `e + r` → `er`
2. `b + e` → `be` (dari kata lain)
3. `a + j` → `aj`
4. `aj + ar` → `ajar`
5. `bel + ajar` → `belajar`

Urutan merge ini disimpan dan digunakan saat tokenisasi teks baru.



Merge tree menunjukkan bagaimana karakter digabungkan secara hierarkis menjadi subword dan kata.

Token Spesial

Selain token dari teks, tokenizer menambahkan **token spesial** dengan fungsi khusus:

Token	Fungsi	Contoh Penggunaan
`<PAD>`	Padding agar panjang batch seragam	Batch processing
`<BOS>`	Beginning of Sequence	Menandai awal teks
`<EOS>`	End of Sequence	Menandai akhir generasi
`<UNK>`	Unknown / token tak dikenal	Fallback untuk OOV
`<MASK>`	Masking token	Masked LM (BERT)
`<SEP>`	Separator antar segment	Pemisah kalimat (BERT)

- **GPT-2:** Hanya menggunakan `<|endoftext|>` sebagai token spesial (minimalis)
- **ChatGPT:** Menambahkan token peran seperti `<|im_start|>system`, `<|im_start|>user`
- Token spesial biasanya mendapat ID tetap di awal vocabulary

Pertimbangan Desain Vocabulary

- **Ukuran vocabulary** mempengaruhi trade-off fundamental:
  - Vocab kecil $\rightarrow$ embedding table kecil, tapi sequence panjang
  - Vocab besar $\rightarrow$ sequence pendek, tapi embedding table besar dan banyak token jarang yang under-trained
- **Ukuran umum di LLM modern:**
  - GPT-2: 50.257 token
  - LLaMA: 32.000 token
  - LLaMA-3: 128.000 token (lebih besar untuk multilingualitas)
  - Gemma: 256.000 token
- **Multibahasa:** Tokenizer yang di-training terutama pada teks Inggris cenderung **tidak efisien** untuk bahasa lain — "Universitas" bisa dipecah menjadi 3–4 subword di tokenizer GPT-2, sementara kata Inggris sepadan hanya 1–2 token.
- **Implikasi biaya:** API LLM menghitung biaya per token — tokenizer yang buruk untuk bahasa Anda berarti biaya lebih mahal!

Dari Encoder-Decoder ke Decoder-Only

- Transformer asli (Vaswani, 2017) memiliki **Encoder** dan **Decoder**:
  - Encoder: bidirectional attention (melihat seluruh input)
  - Decoder: causal/masked attention (hanya melihat token sebelumnya)
  - Dirancang untuk sequence-to-sequence (misal: machine translation)
- **GPT (Radford et al., 2018):** Hanya menggunakan **Decoder stack** — menghilangkan Encoder sepenuhnya.
- **Mengapa?** Language modeling hanya perlu memprediksi token berikutnya:

$$P(w_t \mid w_1, w_2, \ldots, w_{t-1})$$

- Tidak perlu melihat token masa depan — **causal (autoregressive)** sudah cukup.
- **Perbandingan:**
  - **BERT** (Encoder-only): bidirectional, cocok untuk *understanding* (klasifikasi, NER)
  - **GPT** (Decoder-only): unidirectional, cocok untuk *generation* (teks, kode)
  - **T5** (Encoder-Decoder): cocok untuk *seq2seq* (terjemahan, summarization)

Causal Masking: Lower Triangular Matrix

Kunci arsitektur GPT: Setiap token hanya boleh "melihat" dirinya sendiri dan token-token **sebelumnya**.

Ini diimplementasikan dengan **causal mask** pada attention scores:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \mathbf{M}\right) V$$

dimana $\mathbf{M}$ adalah matriks mask **lower triangular**:

$$\mathbf{M} = \begin{pmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- Posisi $-\infty$ setelah softmax menjadi $0 \rightarrow$ attention weight $= 0$ (informasi diblokir)
- Token ke-t hanya bisa attend ke posisi $1, 2, \ldots, t$
- Berbeda dari BERT yang menggunakan **full attention** (semua posisi terlihat)

Autoregressive Generation

GPT menghasilkan teks secara **autoregressive** — satu token per langkah:

Autoregressive Factorization

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, w_2, \ldots, w_{t-1})$$

Proses generasi:

1. Input: "Ibu kota Indonesia adalah" (prompt)
2. Model memprediksi distribusi probabilitas token berikutnya
3. Sampling/greedy: pilih "Jakarta" (probabilitas tertinggi)
4. Append: "Ibu kota Indonesia adalah Jakarta"
5. Ulangi dari langkah 2 sampai <EOS> atau panjang maksimum

Training objective: Minimasi *cross-entropy loss* (next-token prediction):

$$\mathcal{L} = -\sum^{T} \log P(w_t \mid w_{<t};\ \theta)$$

Arsitektur Lengkap GPT

Pipeline end-to-end dari teks mentah ke prediksi:

1. **Tokenization:** Teks $\rightarrow$ token IDs $[t_1, t_2, \ldots, t_n]$
2. **Token Embedding:** $\mathbf{E}_{\text{tok}} \in \mathbb{R}^{|V| \times d}$ — lookup table

$$\mathbf{h}_i^{(0)} = \mathbf{E}_{\text{tok}}[t_i] + \mathbf{E}_{\text{pos}}[i]$$

3. **Position Embedding:** $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{L_{\max} \times d}$ — learnable (bukan sinusoidal)
4. $N\times$ **Transformer Block** (Pre-Norm, dengan causal mask):

$$\mathbf{h}^{(\ell)} = \text{TransformerBlock}^{(\ell)}(\mathbf{h}^{(\ell-1)}) \quad \text{untuk } \ell = 1, \ldots, N$$

5. **Final LayerNorm:** $\mathbf{h}_{\text{final}} = \text{LayerNorm}(\mathbf{h}^{(N)})$
6. **LM Head** (Linear + Softmax):

$$P(w_{t+1} \mid w_{\leq t}) = \text{softmax}(\mathbf{h}_{\text{final},\, t} \cdot \mathbf{E}_{\text{tok}}^T)$$

(Weight tying: LM Head berbagi bobot dengan token embedding!)

GPT vs BERT: Perbandingan Arsitektur

GPT (Decoder-Only)

- **Attention:** Causal (unidirectional) — token hanya melihat ke kiri
- **Training:** Next-token prediction (autoregressive LM)
- **Keunggulan:** Generasi teks alami, scalable, few-shot learning
- **Model:** GPT-2, GPT-3, GPT-4, LLaMA, Mistral

BERT (Encoder-Only)

- **Attention:** Full/bidirectional — token melihat seluruh sequence
- **Training:** Masked Language Modeling (isi token yang di-mask)
- **Keunggulan:** Pemahaman konteks lengkap, representasi kaya
- **Model:** BERT, RoBERTa, DeBERTa, IndoBERT

Tren Saat Ini

Arsitektur **Decoder-Only (GPT-style)** mendominasi era LLM modern. Alasan: (1) pre-training sederhana (next-token prediction), (2) scaling law menguntungkan, (3) kemampuan in-context learning dan generasi teks yang kuat.

Kesimpulan Pertemuan 4

1. **Tokenization** adalah langkah kritis pertama dalam pipeline LLM — mengubah teks menjadi angka. Pilihan strategi mempengaruhi efisiensi dan kualitas model.
2. **Subword tokenization** (BPE, WordPiece, SentencePiece) memberikan keseimbangan optimal antara vocabulary size dan sequence length.
3. **BPE** bekerja dengan iteratif menggabungkan pasangan simbol tersering — sederhana namun sangat efektif.
4. **Special tokens** (`PAD`, `BOS`, `EOS`, `UNK`) memberikan sinyal struktural penting bagi model.
5. **GPT** menggunakan arsitektur **Decoder-Only** dengan *causal masking* untuk autoregressive generation: $P(w_t \mid w_{<t})$.
6. Pipeline GPT: Token Embedding + Position Embedding $\rightarrow$ $N\times$ Transformer Block $\rightarrow$ LM Head.

Pertemuan berikutnya: Training LLM — Pre-training, Fine-tuning, dan Scaling Laws.

Referensi

- Sennrich, R., Haddow, B., & Birch, A. (2016). "Neural Machine Translation of Rare Words with Subword Units." *ACL*.
- Radford, A., et al. (2018). "Improving Language Understanding by Generative Pre-Training." *OpenAI*.
- Radford, A., et al. (2019). "Language Models are Unsupervised Multitask Learners." *OpenAI* (GPT-2).
- Devlin, J., et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *NAACL*.
- Kudo, T. & Richardson, J. (2018). "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing." *EMNLP*.
- Hugging Face Tokenizers Documentation. `https://huggingface.co/docs/tokenizers/`
- Karpathy, A. "Let's build the GPT Tokenizer."