BAB 2
LANDASAN TEORI

2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) merupakan bidang dalam kecerdasan buatan yang berfokus pada interaksi antara komputer dan bahasa manusia, baik dalam bentuk lisan maupun tulisan [29]. Sejak awal kemunculannya pada akhir 1940-an hingga 1960-an, NLP berfokus pada pendekatan simbolik berbasis struktur sintaksis dan statistik seperti frekuensi kata [30]. Namun, pendekatan awal ini memiliki keterbatasan dalam memahami makna karena hanya mengandalkan representasi permukaan teks [29]. Perkembangan lebih lanjut mengarah pada pendekatan semantik untuk menangani ambiguitas makna dalam bahasa alami [30]. Saat ini, NLP digunakan dalam berbagai aplikasi seperti penerjemahan otomatis, analisis sentimen, dan penjawaban pertanyaan [31], dengan fokus pada pemahaman makna, struktur kalimat, serta konteks melalui teknik pembelajaran mesin dan pembelajaran mendalam [32].

2.1.1 ChatGPT

ChatGPT merupakan salah satu implementasi dari model bahasa besar (Large Language Model/LLM) berbasis arsitektur Transformer yang dikembangkan oleh OpenAI, dengan fondasi utama berupa model GPT (Generative Pre-trained Transformer) [10]. Model ini dilatih dengan pendekatan unsupervised learning menggunakan data dalam jumlah besar dari internet, kemudian disempurnakan dengan reinforcement learning from human feedback (RLHF) untuk meningkatkan kualitas dan keamanan respons yang dihasilkan [33]. ChatGPT menunjukkan kemampuan luar biasa dalam menghasilkan teks yang koheren, relevan dengan konteks, dan menyerupai gaya penulisan manusia, sehingga banyak digunakan dalam aplikasi seperti asisten virtual, penulisan otomatis, hingga pendidikan [13]. Namun, meskipun efektif, kemampuan model ini juga menimbulkan tantangan etis dan teknis, seperti risiko penyebaran misinformasi dan kesulitan dalam membedakan teks buatan dengan teks manusia [15].

2.2 Text Preprocessing

Tahap pra-pemrosesan teks bertujuan untuk menyiapkan data agar sesuai dengan format input yang diharapkan sebelum digunakan oleh model machine learning [34]. Proses ini meliputi beberapa teknik umum yang diterapkan pada data teks sebelum dilakukan pelatihan atau inferensi, sebagai berikut:

1. *Data Cleaning*
   Data cleaning merujuk pada langkah-langkah untuk menstandarkan teks serta menghapus elemen yang tidak relevan [35]. Tahap ini melibatkan penghapusan elemen seperti URL, tautan, *hashtag*, *mention*, tanda baca, dan karakter non-alfabet yang diganti dengan spasi [36].

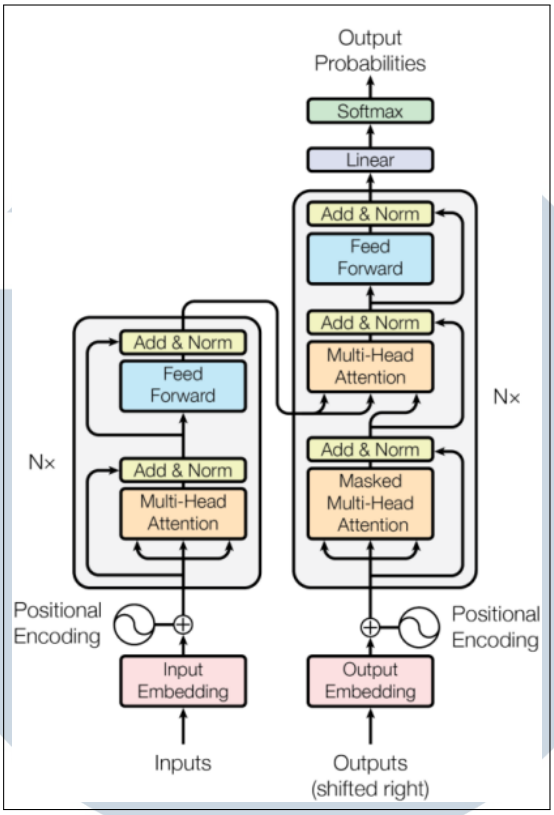
2. *Normalization*
   Normalisasi dilakukan untuk mengubah representasi teks menjadi bentuk standar [37]. Proses ini mencakup normalisasi karakter (seperti konversi Unicode ke ASCII), normalisasi whitespace (menghapus spasi berlebih), dan normalisasi format teks (seperti penanganan newline dan karakter khusus).

3. *Short Text Filtering*
   Konten dengan panjang kurang dari ambang batas tertentu dihapus karena dianggap kurang informatif untuk tugas klasifikasi. Ini membantu menjaga kualitas representasi teks yang akan dipelajari oleh model.

4. *Class Labeling*
   Proses pemberian label pada data teks sesuai dengan kategori yang telah ditentukan. Dalam klasifikasi teks, setiap sampel diberi label yang merepresentasikan kelas atau kategori tertentu. Pelabelan ini menyediakan ground truth yang diperlukan untuk pelatihan model klasifikasi supervised learning.

2.2.1 Model Transformer

Transformer merupakan arsitektur jaringan saraf untuk mengolah data sekuensial yang diperkenalkan sebagai solusi atas keterbatasan model RNN dan LSTM dalam menangkap dependensi jangka panjang. Berbeda dengan model sekuensial, Transformer mengandalkan mekanisme *self-attention* yang memproses seluruh sekuens secara paralel [38].

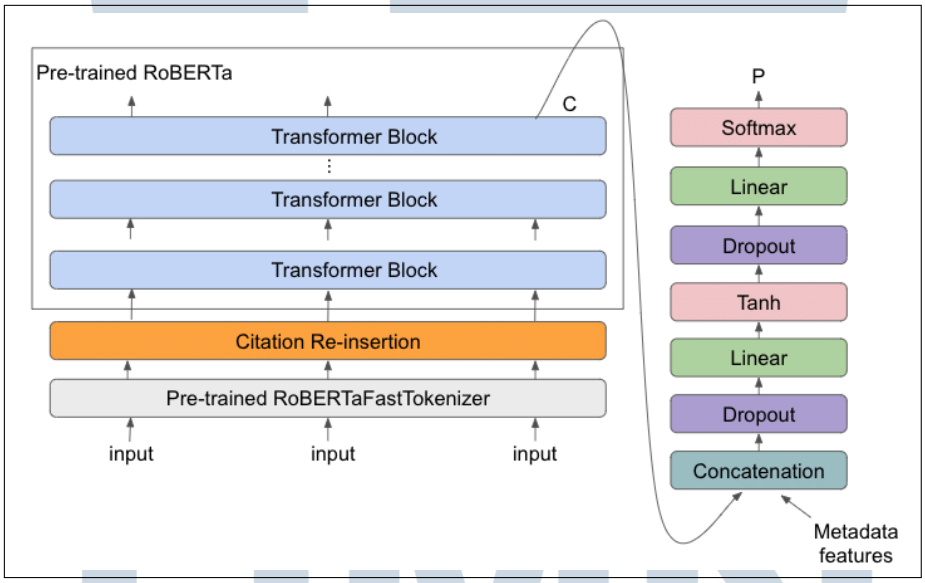


Gambar 2.1. Arsitektur Transformer

Gambar 2.1 menampilkan arsitektur Transformer yang terbagi menjadi bagian encoder di sebelah kiri dan decoder di sebelah kanan. Proses dimulai ketika setiap token diubah menjadi vektor embedding lalu dijumlahkan dengan vektor positional encoding sehingga informasi urutan tetap terjaga. Encoder terdiri atas *N* blok identik. Dalam setiap blok terdapat mekanisme multi-head self-attention yang memungkinkan model mempelajari keterkaitan antar-token secara paralel, diikuti oleh jaringan *feed-forward* dua lapis untuk transformasi non-linier. Kedua sub-lapisan tersebut dibungkus oleh residual connection dan layer normalization guna menjaga stabilitas gradien. Decoder memiliki *N* blok dengan struktur serupa, tetapi diawali oleh masked multi-head self-attention agar prediksi bersifat autoregresif. Setelah itu decoder melakukan attention terhadap keluaran encoder (encoder–decoder attention) sebelum melewati jaringan feed-forward. Setiap sublapisan pun dilengkapi residual connection dan layer normalization. Keluaran blok decoder terakhir diteruskan ke transformasi linear dan fungsi softmax untuk menghasilkan probabilitas token pada kosakata. Desain bebas komponen sekuensial ini memungkinkan pemrosesan paralel dan menjadi dasar bagi model-model seperti BERT, GPT, dan RoBERTa [38].

A RoBERTa

RoBERTa merupakan singkatan dari *A Robustly Optimized BERT Pretraining Approach*, yaitu model representasi bahasa yang dikembangkan oleh Facebook AI Research sebagai pengembangan dari arsitektur BERT. Model ini mempertahankan struktur dasar BERT yang berbasis *encoder Transformer*, namun dengan beberapa modifikasi penting yang meningkatkan kualitas representasi semantiknya. RoBERTa menunjukkan performa superior dibandingkan BERT pada berbagai *benchmark* NLP standar, seperti GLUE, SQuAD, dan RACE, yang mencakup beragam tugas mulai dari klasifikasi, pertanyaan-jawaban, hingga inferensi teks [25, 39]



Gambar 2.2. Arsitektur Model RoBERTa

Gambar 2.2 menampilkan arsitektur RoBERTa untuk klasifikasi teks. Arsitektur ini terdiri dari tiga komponen utama yang bekerja secara berurutan: tokenizer, blok-blok transformer, dan kepala klasifikasi. Keunggulan utama arsitektur ini adalah kemampuannya menangkap hubungan kontekstual dalam teks melalui mekanisme perhatian bidirectional. Komponen pertama adalah *RoBERTaFastTokenizer* yang mengubah teks mentah menjadi token numerik menggunakan algoritma *Byte-Pair Encoding* (BPE). Tokenizer ini memiliki keunggulan dalam:

1. Memecah kata-kata jarang menjadi unit subkata
2. Mempertahankan token utuh untuk kata umum

3. Menangani kata di luar kosakata dengan efektif

Tokenizer menambahkan token khusus seperti [CLS] di awal urutan (untuk representasi seluruh teks) dan [SEP] untuk memisahkan segmen teks, serta menerapkan padding dan truncation untuk mengoptimalkan pemrosesan. Komponen kedua adalah blok-blok *Transformer* yang memproses urutan token menjadi representasi kontekstual. Setiap blok terdiri dari:

1. Mekanisme *multi-head self-attention* yang memungkinkan model fokus pada berbagai bagian teks secara bersamaan dan menangkap hubungan jarak jauh antar token

2. Jaringan neural *feed-forward* yang melakukan transformasi non-linear untuk meningkatkan kapasitas representasi

Blok-blok ini memungkinkan model menangkap pola-pola halus dan hubungan semantik kompleks dalam penggunaan bahasa natural. Komponen ketiga adalah kepala klasifikasi yang mengubah representasi kontekstual menjadi prediksi kelas. Kepala klasifikasi mengambil representasi token [CLS] yang merangkum seluruh teks, kemudian memprosesnya melalui beberapa lapisan neural untuk menghasilkan logit. Kepala klasifikasi ini dapat dikonfigurasi untuk berbagai tugas klasifikasi, baik biner maupun multi-kelas, tergantung pada kebutuhan aplikasi. Terakhir, fungsi *softmax* mengubah logit menjadi probabilitas, seperti ditunjukkan pada *Softmax Layer* dalam gambar. Fungsi ini menghasilkan distribusi probabilitas dengan nilai antara 0 dan 1 yang berjumlah 1. Secara matematis:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.1}$$

Hasil akhirnya adalah distribusi probabilitas yang menunjukkan kecenderungan teks terhadap kelas-kelas yang telah ditentukan. Transformasi linier yang diterapkan pada *Linear Layer* dapat dinyatakan melalui persamaan berikut:

$$y = Wx + b \tag{2.2}$$

Pada Persamaan 2.2, x merepresentasikan vektor input, W adalah matriks bobot yang dipelajari selama proses pelatihan, dan b merupakan vektor bias. Hasil

akhir dari keseluruhan proses ini adalah probabilitas prediksi yang ditunjukkan dengan simbol *P* pada Gambar 2.2.

2.3 Confusion Matrix

Confusion Matrix (CM) adalah metode yang digunakan untuk mengevaluasi kinerja sebuah model dengan menghitung jumlah prediksi yang benar dan salah, kemudian memasukkannya ke dalam sebuah matriks 2x2. Matriks ini selanjutnya dapat digunakan untuk menghitung precision, recall, dan F1-score untuk setiap hasil klasifikasi. Struktur tabel CM dapat dilihat pada Tabel 2.1.

Tabel 2.1. Confusion Matrix

	Predicted Positive	*Predicted Negative*
Actual Positive	*True Positive (TP)*	*False Negative (FN)*
Actual Negative	*False Positive (FP)*	*True Negative (TN)*

Keterangan:

1. *True Positive (TP)*: Jumlah data yang positif dan berhasil diklasifikasikan sebagai positif oleh model.
2. *True Negative (TN)*: Jumlah data yang negatif dan berhasil diklasifikasikan sebagai negatif oleh model.
3. *False Positive (FP)*: Jumlah data yang negatif tetapi diklasifikasikan sebagai positif oleh model.
4. *False Negative (FN)*: Jumlah data yang positif tetapi diklasifikasikan sebagai negatif oleh model.

Setelah didapatkan CM, perhitungan *accuracy*, *precision*, *recall*, dan *F1-score* dapat dilakukan dengan menggunakan rumus-rumus sebagai berikut [40].

1. *Accuracy* mengukur proporsi keseluruhan prediksi yang benar, baik kelas positif maupun negatif, terhadap total data yang diuji. Rumus untuk menghitung nilai accuracy ditunjukkan pada 2.3.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

2. *Precision* mengukur ketepatan model dalam memprediksi kelas positif, yaitu seberapa banyak dari seluruh prediksi positif yang benar-benar positif. Rumus precision ditunjukkan pada 2.4.

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

3. *Recall* mengukur sejauh mana model dapat mengenali seluruh data yang benar-benar positif. Rumus recall dapat dilihat pada 2.5.

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

4. *F1 Score* merupakan rata-rata harmonik dari precision dan recall, yang digunakan untuk menyeimbangkan kedua metrik tersebut terutama ketika distribusi data tidak seimbang. Rumus F1 Score ditunjukkan pada 2.6.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.6}$$