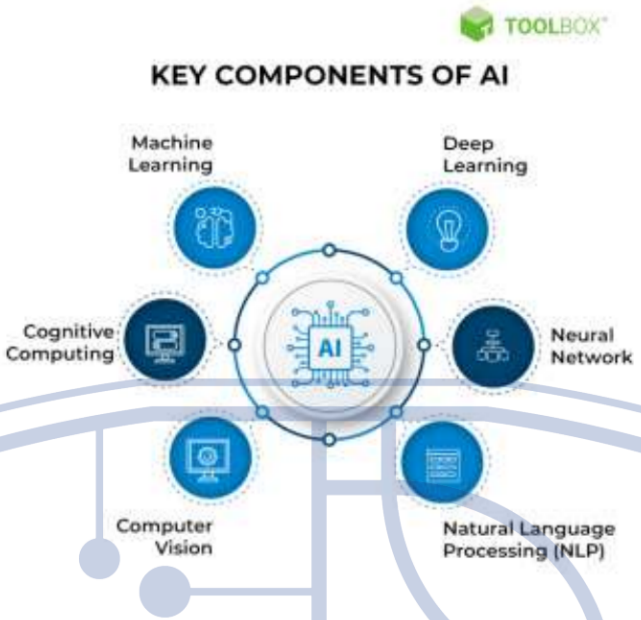
BAB II
KAJIAN LITERATUR

2.1 *Artificial Intelligent* (AI)

Artificial Intelligence (kecerdasan buatan) dapat dipahami sebagai upaya rekayasa sistem atau mesin yang mampu mensimulasikan proses kognitif manusia, seperti mengenali pola, memahami bahasa, belajar dari data, mengambil keputusan, dan merencanakan tindakan. Dalam berbagai bidang, AI digunakan untuk meningkatkan efisiensi, produktivitas, dan mutu pengambilan keputusan, baik di sektor publik maupun industri, termasuk layanan kesehatan, pendidikan, transportasi, dan penelitian ilmiah [18].

Kecerdasan buatan mencakup beberapa komponen utama yang saling melengkapi dalam membentuk kemampuan komputasi yang menyerupai cara berpikir manusia. *Machine learning* (ML) berperan sebagai dasar yang memungkinkan sistem belajar dari data dan menyesuaikan perilaku berdasarkan pola yang telah dipelajari. Dari pendekatan ini berkembang *deep learning* (DL) yang memanfaatkan struktur jaringan saraf tiruan berlapis (*artificial neural network*) layaknya neuron pada otak manusia untuk memproses informasi yang lebih kompleks. Kemampuan memahami bahasa diwujudkan melalui *natural language processing* (NLP), sedangkan kemampuan mengenali dunia visual dikembangkan melalui *computer vision* yang memungkinkan sistem menginterpretasikan gambar dan objek. Sementara itu, *cognitive computing* berupaya mengintegrasikan seluruh kemampuan tersebut agar sistem AI dapat menalar, memahami konteks, dan mengambil keputusan secara lebih menyerupai cara berpikir manusia [19].

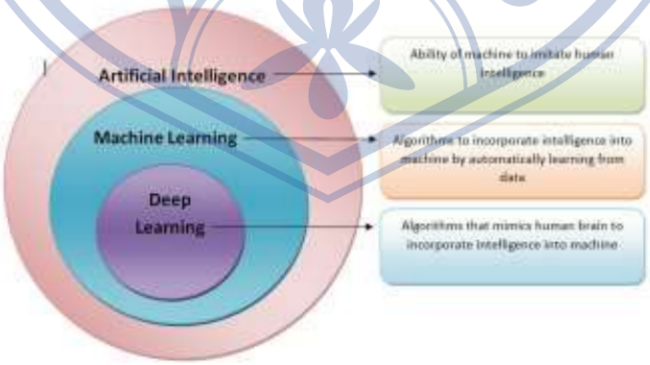
Kecerdasan buatan menggambarkan langkah besar dalam evolusi teknologi, di mana sistem komputer tidak lagi sekadar menjalankan perintah, tetapi mampu memahami konteks dan beradaptasi terhadap informasi yang terus berubah. Perkembangannya menandai pergeseran menuju era komputasi yang lebih cerdas, dinamis, dan berorientasi pada pemecahan masalah kompleks [18].



Gambar 2.1 Komponen Kunci dalam AI [20]

2.1.1 *Machine Learning*

Machine learning (ML) adalah cabang fundamental dari *Artificial Intelligence* (AI) yang berfokus pada pengembangan algoritma yang memungkinkan sistem komputer untuk belajar secara mandiri dari data dan meningkatkan kinerjanya seiring waktu tanpa diprogram secara eksplisit untuk setiap tugas [21]. Inti dari ML adalah kemampuannya untuk mengidentifikasi pola-pola kompleks dari kumpulan data besar, lalu menggunakan pola tersebut untuk membuat prediksi atau keputusan yang akurat. Proses ini secara fundamental mengubah cara komputasi bekerja, beralih dari instruksi eksplisit ke pembelajaran berbasis pola dari pengalaman [15,16].



Gambar 2.2 Hierarki Konsep Kecerdasan Buatan [23]

Proses pembelajaran dalam ML dimulai dengan pemberian *training data* kepada sebuah algoritma. Algoritma ini kemudian membangun sebuah model matematis berdasarkan data

tersebut. Kualitas model yang dihasilkan sangat bergantung pada kualitas dan kuantitas data yang digunakan. Secara umum, teknik *machine learning* diklasifikasikan menjadi empat kategori utama, yaitu *supervised learning*, *unsupervised learning*, *semi-supervised learning*, dan *reinforcement learning* [22].

1. *Supervised Learning* (Pembelajaran Terarah) adalah pendekatan di mana model dilatih menggunakan *dataset* yang telah diberi label, artinya setiap data *input* sudah memiliki pasangan *output* yang benar. Metode ini sangat efektif untuk tugas klasifikasi (misalnya, mendiagnosis penyakit berdasarkan citra medis) dan regresi (misalnya, memprediksi tren pasar saham). Algoritma populer dalam kategori ini termasuk *Support Vector Machine* (SVM), *Decision Tree*, dan *Naïve Bayes* [15,16].
2. *Unsupervised Learning* (Pembelajaran Tak Terarah) digunakan pada data yang tidak memiliki label. Tujuannya adalah untuk menemukan struktur atau pola tersembunyi di dalam data itu sendiri. Teknik seperti *k-Means clustering*, *Gaussian Mixture Model* (GMM) dan lainnya adalah teknik yang biasa digunakan untuk *clustering* (pengelompokan data), estimasi kepadatan, dan reduksi dimensi [15,16].
3. *Semi-supervised Learning* (Pembelajaran Semi-Terarah) merupakan gabungan antara *supervised* dan *unsupervised learning*, karena beroperasi pada data yang sebagian berlabel dan sebagian tidak berlabel. Pendekatan ini sangat berguna dalam skenario dunia nyata di mana data berlabel sulit atau mahal untuk diperoleh, sementara data tidak berlabel tersedia dalam jumlah besar [22].
4. *Reinforcement Learning* (Pembelajaran Penguatan) merupakan metode pembelajaran di mana sistem belajar mengevaluasi dan memilih tindakan yang paling optimal untuk memaksimalkan hasil atau efisiensi melalui proses interaksi dengan lingkungannya. Pembelajaran ini didasarkan pada konsep *reward* (hadiah) dan *penalty* (hukuman) sebagai umpan balik atas setiap tindakan yang dilakukan. Pendekatan ini terbukti efektif untuk melatih model kecerdasan buatan pada sistem yang kompleks, seperti robotika, transportasi otomatis (*self-driving*), dan manajemen logistik [22].

2.1.2 *Deep Learning*

Deep Learning (DL) adalah bagian dari metode *machine learning* yang lebih luas dan merupakan salah satu paradigma komputasi yang dianggap sebagai standar emas dalam komunitas ML saat ini. DL secara spesifik menggunakan arsitektur *Artificial Neural Network*

(ANN) dengan banyak lapisan (*multi-layered*) untuk mempelajari dan mengenali pola-pola rumit dari *dataset* berskala besar [24]

Perbedaan mendasar antara DL dan ML konvensional terletak pada proses rekayasa fitur (*feature engineering*). Pada ML tradisional, proses ekstraksi dan seleksi fitur yang relevan dari data mentah seringkali memerlukan intervensi manusia dan sangat spesifik untuk domain tertentu. Sebaliknya, algoritma DL memiliki kemampuan untuk mengekstraksi fitur secara otomatis dan hierarkis. Lapisan-lapisan awal dalam arsitektur DL akan mengekstraksi fitur-fitur tingkat rendah, sementara lapisan-lapisan yang lebih dalam akan membangun fitur-fitur tingkat tinggi yang lebih kompleks dan abstrak. Kemampuan belajar representasi secara otomatis inilah yang membuat DL sangat kuat, terutama saat berhadapan dengan *big data* [24].

Keberhasilan *deep learning* tidak hanya ditentukan oleh kemampuannya mengekstraksi fitur secara otomatis, tetapi juga oleh kemajuan ekosistem teknologi pendukungnya. Ketersediaan data dalam jumlah besar (*big data*), peningkatan kapasitas komputasi melalui *Graphics Processing Unit* (GPU), serta pengembangan berbagai *open-source framework* seperti *TensorFlow* dan *PyTorch* telah mempercepat penerapannya di berbagai bidang. Pendekatan ini memungkinkan pengembangan model yang lebih adaptif dan dapat digeneralisasi lintas domain, menjadikannya salah satu pendorong utama kemajuan kecerdasan buatan modern [24].

Beberapa contoh arsitektur *deep learning* yang sering digunakan antara lain sebagai berikut:

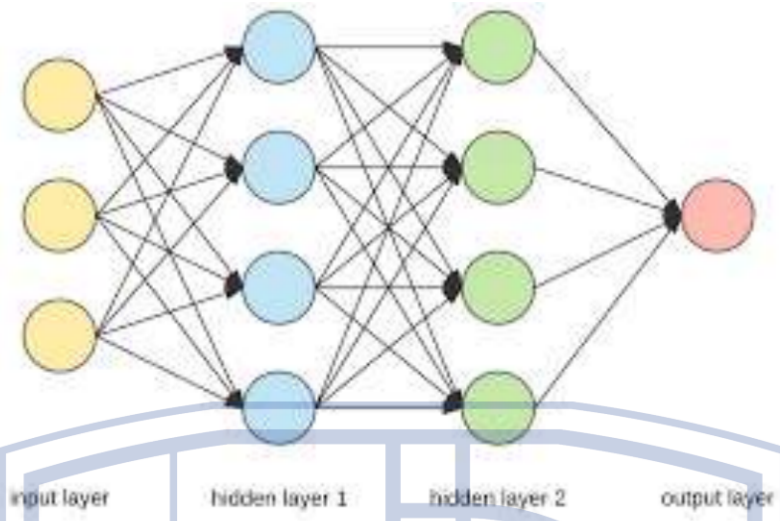
1. *Artificial Neural Network* (ANN) merupakan arsitektur dasar dari *deep learning* yang tersusun atas beberapa lapisan neuron. ANN bekerja dengan meniru cara kerja otak manusia dalam mengenali pola melalui proses pelatihan berbasis *backpropagation* yang berfungsi untuk menyesuaikan bobot jaringan berdasarkan selisih antara prediksi dan target. Arsitektur ini digunakan secara luas untuk tugas klasifikasi, prediksi, dan pengenalan pola [24].
2. *Convolutional Neural Network* (CNN) merupakan arsitektur yang sangat populer dan sering digunakan, terutama untuk tugas-tugas yang melibatkan data dengan topologi seperti *grid*, misalnya gambar. CNN secara otomatis dapat mendeteksi fitur-fitur penting tanpa pengawasan manusia, menjadikannya sangat kuat untuk analisis citra medis, pengenalan objek, dan klasifikasi gambar [18,19].
3. *Recurrent Neural Network* (RNN) merupakan arsitektur yang dirancang khusus untuk memproses data sekuensial atau deret waktu, seperti teks atau data audio percakapan. Setiap unit jaringan memiliki *hidden state* yang berfungsi menyimpan informasi dari

langkah sebelumnya dan meneruskannya ke langkah berikutnya, sehingga hubungan antarurutan dapat dipelajari. Namun, RNN konvensional sering mengalami masalah *vanishing gradient*, yaitu kondisi ketika nilai gradien mengecil drastis selama proses pelatihan, sehingga model sulit mempelajari hubungan jangka panjang [24].

4. *Long Short-Term Memory* (LSTM) adalah arsitektur jaringan saraf yang dirancang untuk mempelajari pola berurutan dan menjaga informasi penting dalam rentang waktu yang panjang. Model ini mengatasi masalah *vanishing gradient* melalui penggunaan mekanisme gerbang (*input gate*, *forget gate*, dan *output gate*) yang mengatur aliran informasi secara adaptif. Dengan struktur tersebut, LSTM mampu mempertahankan konteks yang relevan dan menghasilkan representasi yang lebih stabil untuk data sekuensial [26].
5. *Gated Recurrent Unit* (GRU) merupakan arsitektur jaringan saraf yang memiliki struktur lebih sederhana namun tetap efektif dalam mempertahankan informasi jangka panjang. Model ini menyederhanakan mekanisme LSTM dengan menggabungkan *input gate* dan *forget gate* menjadi satu gerbang pembaruan (*update gate*), serta menambahkan *reset gate* untuk mengontrol seberapa banyak informasi masa lalu yang dipertahankan. Struktur yang lebih ringkas ini membuat proses pelatihan lebih cepat dan efisien tanpa mengorbankan kemampuan model dalam memahami konteks pada data teks atau deret waktu [26].

2.2 *Artificial Neural Network (ANN)*

Artificial Neural Network (ANN) atau Jaringan Saraf Tiruan adalah model komputasi yang terinspirasi oleh struktur dan fungsi otak manusia. ANN terdiri dari tiga jenis lapisan *node* yang saling terhubung *input layer*, satu atau lebih *hidden layers*, dan *output layer* [21]. Kemampuan utamanya adalah memodelkan hubungan nonlinier yang kompleks dan belajar langsung dari data, menjadikannya fondasi bagi sebagian besar arsitektur *deep learning* [27].



Gambar 2.3 Struktur *Feedforward Neural Network* [28]

Dibandingkan dengan metode statistik konvensional, ANN memiliki keunggulan dalam mengenali pola dan memproses data secara adaptif. Keunggulan ini dicapai melalui algoritma pelatihan *backpropagation*, di mana jaringan secara iteratif menyesuaikan bobot (*weights*) untuk meminimalkan kesalahan antara hasil prediksi dan nilai sebenarnya. Pendekatan ini memungkinkan ANN belajar langsung dari data dan meningkatkan akurasi seiring bertambahnya jumlah pelatihan [27].

2.2.1 *Perceptron* dan Fungsi Aktivasi

Proses pembelajaran pada *Artificial Neural Network* (ANN) berawal dari unit dasar yang disebut *perceptron*. Setiap *perceptron* menerima sejumlah masukan (*input*), mengalikan masing-masing dengan bobot tertentu, menjumlahkannya, lalu menambahkan bias. Nilai total ini kemudian dilewatkan melalui fungsi aktivasi untuk menghasilkan keluaran (*output*). Mekanisme ini memungkinkan satu neuron belajar melakukan keputusan sederhana, seperti mengklasifikasikan pola berdasarkan ambang tertentu [29].

Agar model mampu mempelajari pola yang lebih kompleks, banyak neuron digabungkan menjadi satu lapisan. Lapisan-lapisan tersebut kemudian dihubungkan secara berurutan dari lapisan masukan (*input layer*), beberapa lapisan tersembunyi (*hidden layers*), hingga lapisan keluaran (*output layer*). Setiap lapisan menerima hasil aktivasi dari lapisan sebelumnya dan mengubahnya menjadi representasi baru melalui fungsi aktivasi non-linier [29]. Struktur berlapis inilah yang memungkinkan jaringan saraf menangkap hubungan non-linier dan membangun abstraksi fitur secara bertahap [30].

Beragam fungsi aktivasi digunakan sesuai karakteristik tugas dan rentang nilai yang diinginkan. *ReLU* (*Rectified Linear Unit*) menghasilkan nol untuk nilai negatif dan

mempertahankan nilai positif sebagaimana adanya, sehingga sederhana dan cepat dalam pelatihan. *Sigmoid* memetakan keluaran ke rentang 0 sampai 1 dan sering dipakai pada klasifikasi biner. *Tanh* mengubah nilai ke rentang –1 hingga 1 dan menjaga keseimbangan data di sekitar nol. *Softmax* mengubah sekumpulan nilai menjadi distribusi probabilitas yang menjumlah ke satu, umum digunakan pada klasifikasi multi-kelas [31]. Secara sistematis, beberapa fungsi aktivasi fundamental tersebut didefinisikan sebagai berikut [31,32]:

1. *ReLU*, fungsi aktivasi yang mengubah semua nilai negatif menjadi nol.

$$f(x) = \max(0, x) \quad \text{..... (1)}$$

2. *Sigmoid*, fungsi aktivasi yang memetakan *output* ke rentang antara 0 dan 1, sering digunakan untuk probabilitas.

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{..... (2)}$$

3. *Tanh*, fungsi aktivasi yang memetakan *output* ke rentang antara -1 dan 1.

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \text{..... (3)}$$

4. *Softmax*, fungsi aktivasi yang mengubah vektor nilai mentah menjadi distribusi probabilitas untuk klasifikasi multi-kelas.

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{..... (4)}$$

Selain fungsi aktivasi, setiap neuron memiliki bias yang berfungsi menggeser ambang keputusan agar model lebih fleksibel dan tidak terkunci di sekitar nilai nol [19]. Ketika banyak neuron dalam satu lapisan saling terhubung penuh dengan neuron pada lapisan berikutnya, terbentuklah lapisan terhubung penuh (*fully connected layer* atau *dense layer*). Pada tahap ini, setiap neuron menggabungkan semua hasil aktivasi dari lapisan sebelumnya sebelum menerapkan fungsi aktivasi baru, membentuk transformasi non-linier yang lebih kompleks [19]. Secara matematis, hubungan antar lapisan dinyatakan sebagai berikut [33]:

$$z_l = W_l \times a_{l-1} + b_1 \quad \text{..... (5)}$$

2.2.2 Regularisasi, Fungsi *Loss* dan *Optimizer*

Dalam proses pelatihan *Artificial Neural Network* (ANN), model perlu menyeimbangkan kemampuan untuk belajar dari data dan kemampuannya melakukan generalisasi. Ketika jaringan terlalu kompleks atau berlatih terlalu lama pada data terbatas, model dapat mengalami *overfitting*, yaitu kondisi di mana model mengingat pola data latih tetapi gagal memprediksi data baru dengan baik [34].

Regularisasi merupakan serangkaian teknik yang digunakan untuk mengendalikan kompleksitas model agar tidak terlalu menyesuaikan diri terhadap data pelatihan. Tujuannya adalah mencegah *overfitting* dengan menambahkan batasan atau penalti terhadap bobot, atau dengan cara mengurangi ketergantungan antar-neuron. Metode regularisasi umum mencakup penalti L1/L2, *early stopping*, dan *dropout* [35].

Regularisasi L1 menambahkan penalti berdasarkan jumlah absolut bobot, sehingga mendorong sebagian bobot menjadi nol dan menghasilkan model yang lebih sederhana. Regularisasi L2 memberikan penalti terhadap kuadrat nilai bobot, yang membuat bobot besar cenderung dikecilkan secara halus dan menjaga stabilitas pembelajaran. Selain itu, *early stopping* menghentikan proses pelatihan secara otomatis ketika performa model pada data validasi tidak lagi membaik, guna mencegah model belajar berlebihan terhadap data latih. Adapun *dropout* bekerja dengan menonaktifkan sebagian neuron secara acak selama pelatihan untuk mengurangi ketergantungan antarunit dan mencegah *overfitting*. Keempat pendekatan ini berfungsi menyeimbangkan kemampuan belajar model agar tetap akurat tanpa kehilangan daya generalisasi terhadap data baru [36].

Pelatihan jaringan saraf dilakukan menggunakan mekanisme *backpropagation,* yaitu proses perhitungan balik yang digunakan untuk memperbarui bobot jaringan berdasarkan nilai kesalahan (*error*) yang dihasilkan. Dalam tahap ini, *error* yang muncul di lapisan keluaran dihitung dan disebarkan kembali ke lapisan-lapisan sebelumnya untuk menentukan kontribusi setiap bobot terhadap kesalahan tersebut. Nilai gradien dari setiap parameter kemudian digunakan untuk memperbaiki bobot secara iteratif. Dengan cara ini, jaringan secara bertahap meminimalkan kesalahan prediksi dan memperkuat pola yang relevan dalam data pelatihan [37].

Proses *backpropagation* sangat bergantung pada fungsi *loss*, yang berperan sebagai ukuran seberapa besar perbedaan antara hasil prediksi dan nilai target sebenarnya. Salah satu fungsi *loss* yang paling sering digunakan pada klasifikasi biner adalah *Binary Cross-Entropy (BCE)*, yang mengukur jarak antara distribusi probabilitas keluaran model dan label sebenarnya [38]. Secara matematis, fungsi BCE dinyatakan sebagai berikut [39]:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \quad \text{.....(6)}$$

Nilai *loss* yang dihasilkan menjadi acuan bagi algoritma *optimizer*, yang bertugas memperbarui bobot model agar nilai *loss* semakin kecil. *Optimizer* memanfaatkan informasi gradien dari proses *backpropagation* untuk menentukan arah dan besarnya perubahan bobot di setiap iterasi [25].

Salah satu algoritma optimisasi yang paling banyak digunakan adalah *Adam* (*Adaptive Moment Estimation*). Algoritma ini menggabungkan keunggulan *momentum* dan *adaptive learning rate*, sehingga pembaruan bobot menjadi lebih efisien dan stabil, bahkan pada data berukuran besar dan gradien yang fluktuatif. Mekanisme *Adam* pada setiap iterasi pelatihan dapat dirinci melalui langkah-langkah matematis berikut [40]:

1. Hitung Gradien

$$g_t = \nabla_\theta J(\theta_{t-1}) \quad (7)$$

2. Perbarui Estimasi Momen Pertama (Momentum)

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad (8)$$

3. Perbarui Estimasi Momen Kedua (*Adaptive Learning Rate*)

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad (9)$$

4. Koreksi Bias (*Bias Correction*)

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t} \quad (10)$$

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t} \quad (11)$$

5. Perbarui Parameter Model

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon} \quad (12)$$

Algoritma AdamW (*Adaptive Moment Estimation with Decoupled Weight Decay*) merupakan pengembangan dari Adam yang mengatasi kelemahan penerapan regularisasi L2 pada *optimizer* adaptif dengan memisahkan (*decoupling*) mekanisme *weight decay* dari pembaruan gradien, sehingga regularisasi parameter tidak lagi memengaruhi perhitungan *adaptive learning rate* dan mampu meningkatkan kemampuan generalisasi model secara signifikan. Pada AdamW, langkah pembaruan parameternya menjadi [41]:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon} - \alpha \cdot \lambda\, \theta_{t-1} \quad (13)$$

Stabilitas proses optimisasi juga dipengaruhi oleh pengaturan laju belajar (*learning rate scheduling*) dan pengendalian gradien. Misalnya, *ReduceLROnPlateau* menurunkan laju belajar ketika metrik validasi tidak menunjukkan peningkatan, sedangkan teknik *gradient clipping*, seperti parameter *clipnorm* pada kelas *optimizer*, digunakan untuk membatasi besaran gradien agar pembaruan tetap stabil [42].

2.3 *Natural Language Processing* (NLP)

Natural Language Processing (NLP) merupakan bidang dalam kecerdasan buatan yang berfokus pada bagaimana komputer dapat memahami, memproses, dan menghasilkan bahasa

manusia secara alami. Tujuan utama dari NLP adalah penghubung komunikasi antara manusia dan mesin dengan memungkinkan sistem komputer menafsirkan teks atau ucapan layaknya manusia. Teknologi ini telah menjadi dasar bagi berbagai aplikasi modern seperti penerjemahan otomatis, pencarian informasi, asisten virtual, sistem tanya jawab, serta pendeteksian kesamaan semantik antar teks. Dalam penelitian ini, NLP berperan penting dalam mengubah bahasa alami menjadi representasi numerik yang dapat dipelajari oleh model *deep learning* [43].

Secara umum, NLP menggabungkan konsep linguistik dan pembelajaran mesin untuk memahami makna di balik bahasa. Bahasa manusia bersifat ambigu dan kontekstual, sehingga sistem NLP harus mampu menganalisis struktur sintaksis (tata bahasa) dan semantik secara bersamaan [44]. Proses ini melibatkan berbagai tahapan mulai dari pembersihan teks, segmentasi, pengenalan entitas, analisis sintaksis, hingga pemahaman konteks kalimat [45]. Kemajuan algoritma *deep learning* dan representasi vektor seperti *word embedding* telah mempercepat perkembangan NLP, memungkinkan model memahami hubungan semantik antar kata dan menghasilkan representasi yang lebih kaya terhadap konteks kalimat [46].

Untuk mencapai pemahaman tersebut, teks mentah harus melalui serangkaian proses pengolahan agar dapat dikenali oleh model. Tahapan utama dalam NLP meliputi *preprocessing* untuk membersihkan dan menormalkan data, tokenisasi untuk memecah kalimat menjadi unit-unit dasar, dan *word embedding* untuk merepresentasikan kata dalam bentuk vektor numerik [45].

2.3.1 *Preprocessing*

Tahap awal dalam pemrosesan bahasa alami adalah *preprocessing*, yaitu proses membersihkan dan menormalkan teks agar siap diproses oleh algoritma. Data teks mentah umumnya masih mengandung elemen-elemen yang tidak relevan seperti tanda baca, angka, URL, simbol, serta variasi penulisan huruf besar dan kecil yang dapat mengganggu proses pembelajaran model [47]. Tujuan *preprocessing* adalah mengubah teks menjadi format yang konsisten, terstandar, dan informatif [48].

Langkah umum dalam tahap ini meliputi *case folding* (mengubah semua huruf menjadi kecil), penghapusan tanda baca dan karakter khusus, pembersihan *stopwords* yang tidak membawa makna penting (seperti "*the*", "*is*", "*and*"), serta normalisasi teks agar bentuk kata sejenis dapat diseragamkan. Dalam beberapa kasus, juga dilakukan proses *stemming* atau *lemmatization* untuk mengembalikan kata ke bentuk dasarnya. Melalui tahapan ini, data teks

menjadi lebih bersih dan terstruktur sehingga meminimalkan *noise* yang dapat mengganggu proses pembelajaran model [45].

2.3.2 Tokenisasi

Tokenisasi merupakan proses memecah teks menjadi unit-unit terkecil yang disebut *token*, seperti kata, sub-kata, atau karakter. Proses ini penting karena model pembelajaran mesin tidak dapat memproses teks mentah secara langsung, melainkan memerlukan input dalam bentuk satuan yang terdefinisi [49]. Metode tokenisasi tradisional biasanya memisahkan teks berdasarkan spasi atau tanda baca, namun pendekatan modern menggunakan metode yang lebih adaptif seperti *WordPiece* [50].

Metode *WordPiece* memecah kata menjadi potongan sub-kata agar dapat menangani kata baru atau langka yang tidak terdapat dalam kosakata pelatihan (*out-of-vocabulary words*) [51]. Misalnya, kata "unhappiness" dapat dipecah menjadi "un", "##happi", dan "##ness", di mana tanda "##" menunjukkan bahwa token merupakan lanjutan dari sub-kata sebelumnya. Pendekatan ini membuat model lebih efisien dalam menyimpan kosakata dan mampu menangani berbagai variasi bentuk kata tanpa kehilangan makna dasarnya [52].

2.3.3 *Word Embedding*

Word embedding adalah representasi numerik dari kata atau token dalam ruang vektor berdimensi nyata. Setiap kata diubah menjadi vektor angka yang merepresentasikan makna semantiknya, sehingga kata-kata dengan makna serupa akan berada berdekatan dalam ruang vektor tersebut. Pendekatan ini memungkinkan model untuk memahami hubungan semantik antar kata, seperti sinonim atau asosiasi kontekstual [53].

Beberapa metode embedding yang banyak digunakan antara lain *Word2Vec*, *GloVe*, dan *FastText*. Meskipun efektif, metode-metode ini menghasilkan *embedding* yang bersifat statis, yaitu setiap kata memiliki representasi tunggal tanpa mempertimbangkan konteks kalimat. Sebagai contoh, kata "*bank*" akan memiliki vektor yang sama baik dalam kalimat "*I went to the bank to withdraw money*" maupun "*The river bank is flooded*", padahal maknanya berbeda. Keterbatasan inilah yang mendorong pengembangan model berbasis konteks seperti BERT, yang mampu menghasilkan *embedding* dinamis sesuai konteks kemunculan kata [54].

2.4 Pengukuran Kesamaan Teks

Pengukuran kesamaan teks (*textual similarity*) merupakan salah satu komponen penting dalam berbagai aplikasi pemrosesan bahasa. Tujuan utamanya adalah menentukan sejauh mana dua potongan teks baik berupa kata, kalimat, paragraf, maupun dokumen dapat dianggap memiliki makna yang serupa. Proses ini digunakan secara luas dalam berbagai bidang, seperti penilaian jawaban otomatis, pencarian informasi, sistem tanya jawab, deteksi plagiarisme, serta analisis duplikasi dokumen [55]. Dalam *Natural Language Processing (NLP)*, pengukuran kesamaan teks berfungsi sebagai dasar untuk menilai tingkat kesetaraan semantik antara dua representasi bahasa yang berbeda namun mungkin memiliki makna yang sama [56].

Secara prinsip, pengukuran kesamaan teks dilakukan dengan dua tahap utama, yaitu representasi teks dan perhitungan ukuran kemiripan. Teks yang semula berupa data simbolik harus terlebih dahulu dikonversi menjadi bentuk numerik, misalnya dalam bentuk vektor fitur, matriks token, atau *embedding* yang mewakili setiap kata dalam ruang berdimensi tinggi. Setelah representasi numerik diperoleh, kemiripan antarvektor tersebut dapat dihitung menggunakan ukuran jarak atau sudut tertentu, seperti *cosine similarity* atau *dot product*. Pendekatan ini dapat dikelompokkan menjadi dua kategori besar, yaitu pendekatan leksikal dan pendekatan semantik [55].

Pendekatan leksikal menilai kemiripan berdasarkan bentuk permukaan teks, seperti kesamaan kata atau token, sedangkan pendekatan semantik berfokus pada kesamaan makna yang terkandung di dalamnya. Masing-masing pendekatan memiliki keunggulan dan keterbatasan, tergantung pada kebutuhan aplikasi dan kedalaman analisis yang diinginkan [55]. Selain itu, hasil perbandingan representasi teks juga dapat diukur menggunakan berbagai fungsi kemiripan, baik berbasis geometri seperti *cosine similarity* maupun probabilistik melalui fungsi *sigmoid* pada model pembelajaran mendalam [57].

Tabel 2.1 Perbandingan Model STS Pada *Dataset Quora Question Pairs* Berdasarkan Penelitian Sebelumnya

Model	**Acc (%)**	**Prec (%)**	**Rec (%)**	**F1 (%)**
LogReg + TF-IDF [58]	72,56	68,98	46,63	55,65
Random Forest [58]	70,00	59,57	58,28	58,92
XGBoost [58]	71,03	60,51	61,96	61,22

MaLSTM (*Siamese* LSTM) [58]	75,41	68,43	62,00	65,06
Bert-Base-Uncased (*Fine-Tuned*) [58]	86,26	80,08	83,55	81,78
HBAM (*Word2Vec* + BiLSTM + *Attention*) [59]	81,20	78,87	84,93	81,79
BERT-BiGRU [59]	83,30	86,37	77,02	81,43
BBASM (BERT + BiLSTM + *Attention*) [59]	84,45	81,96	87,44	84,61
MLP [60]	72,63	58,78	72,45	64,90
CNN [60]	80,27	71,20	73,49	72,23
LSTM [60]	81,07	68,62	84,41	75,70
LSTM + CNN [60]	81,05	70,04	79,94	74,66
Hybrid Siamese and Feedforward [61]	76,72	-	-	-

2.4.1 Pendekatan Leksikal

Pendekatan leksikal menilai kesamaan teks berdasarkan bentuk permukaan atau struktur literal dari teks tersebut. Fokus utamanya adalah membandingkan elemen-elemen eksplisit seperti huruf, kata, atau urutan token tanpa memperhitungkan makna yang terkandung di dalamnya. Dua teks dianggap mirip apabila memiliki tingkat tumpang-tindih (*overlap*) yang tinggi atau perbedaan karakter yang kecil. Metode-metode klasik yang termasuk dalam pendekatan ini antara lain *string-based* seperti *Levenshtein distance* dan *token-based* seperti *Jaccard coefficient* serta *Dice's coefficient* [62].

Metode *Levenshtein distance* menghitung jumlah minimum operasi yang dibutuhkan untuk mengubah satu teks menjadi teks lain (melalui penambahan, penghapusan, atau penggantian karakter), sedangkan *Jaccard coefficient* menilai rasio antara jumlah irisan dan

gabungan token pada dua teks. *Dice's coefficient* bekerja serupa namun memberi bobot dua kali lipat pada irisan token untuk memperkuat pengaruh kesamaan [62].

Selain metode berbasis *string* dan token, pendekatan leksikal juga dapat menggunakan model berbasis bobot seperti TF-IDF (*Term Frequency–Inverse Document Frequency*). Pada metode ini, setiap kata diubah menjadi nilai numerik yang menunjukkan seberapa penting kata tersebut dalam suatu teks dibandingkan dengan keseluruhan kumpulan teks yang dianalisis. Ukuran kemiripan antarvektor biasanya dihitung dengan *cosine similarity*, yang menilai seberapa searah dua vektor representasi tersebut. Pendekatan leksikal relatif cepat dan efisien secara komputasi, sehingga cocok untuk *dataset* berukuran besar. Namun, karena berfokus pada kesamaan bentuk, pendekatan ini sering gagal menangkap kemiripan makna ketika dua kalimat memiliki struktur berbeda atau menggunakan sinonim, sehingga tidak efektif untuk mendeteksi parafrase atau kesetaraan semantik yang lebih dalam [62].

2.4.2 Pendekatan Semantik

Pendekatan semantik bekerja dengan menilai kesamaan makna berdasarkan konteks dan relasi antar konsep dalam teks, bukan hanya kesamaan kata yang muncul. Dalam pendekatan ini, sistem berupaya memahami hubungan logis antara ide atau entitas dalam kalimat, seperti sinonimi, parafrase, atau keterkaitan sebab-akibat. Dengan memahami struktur semantik ini, model dapat mengenali bahwa dua kalimat berbeda secara leksikal dapat menyampaikan makna yang sama. Misalnya, "*The car was bought by John*" dan "*John purchased the car*" memiliki struktur sintaksis berbeda, tetapi secara semantik identik. Pendekatan ini banyak diterapkan dalam berbagai tugas NLP, termasuk *semantic textual similarity* (STS), karena mampu menangkap kesetaraan makna lintas variasi kalimat [63].

Dibandingkan metode leksikal, pendekatan semantik menawarkan pemahaman yang lebih dalam terhadap konteks bahasa. Model yang menggunakan analisis semantik mampu mengenali hubungan makna yang tidak eksplisit, seperti inferensi atau kesetaraan logis antar pernyataan. Hal ini membuat pendekatan semantik lebih andal dalam menangani kasus parafrase, penerjemahan, serta deteksi plagiarisme berbasis makna. Namun, kompleksitas pemrosesan dan kebutuhan data kontekstual yang besar membuat pendekatan ini lebih menuntut secara komputasi dibanding pendekatan leksikal. Meskipun demikian, hasil yang diperoleh jauh lebih representatif terhadap makna sebenarnya, menjadikannya dasar utama bagi sistem modern dalam mengukur kesamaan teks secara *semantic* [64].

2.4.3 Fungsi Pengukuran Kemiripan

Fungsi pengukuran kemiripan merupakan mekanisme matematis yang digunakan untuk menilai seberapa dekat dua representasi teks satu sama lain. Dalam *Semantic Textual Similarity* (STS), fungsi ini berperan penting untuk mengubah representasi numerik hasil pemodelan menjadi nilai yang dapat diinterpretasikan sebagai tingkat kesamaan makna antar teks. Semakin tinggi nilainya, semakin besar pula kemungkinan bahwa dua kalimat memiliki kesetaraan semantik [63]. Fungsi pengukuran kemiripan yang paling banyak digunakan adalah *cosine similarity*, yang menghitung sudut antara dua vektor dalam ruang berdimensi tinggi. Nilai *cosine* berkisar antara 0 hingga 1, di mana nilai mendekati 1 menunjukkan arah vektor yang hampir sama atau tingkat kemiripan yang tinggi [65]. Selain *cosine similarity*, fungsi pengukuran lain seperti *dot product* juga digunakan, dengan menghitung hasil kali skalar antara dua vektor yang memperhitungkan besar nilai vektor selain arahnya. Kedua pendekatan ini banyak diterapkan baik pada model berbasis TF-IDF maupun pada model *embedding* modern, karena mampu merepresentasikan hubungan antar-representasi teks secara efektif dan komputasinya relatif efisien [63].

Dalam pendekatan berbasis jaringan saraf, perhitungan kesamaan tidak selalu dilakukan secara eksplisit menggunakan fungsi geometrik. Model sering kali mempelajari fungsi kesamaan secara *end-to-end* melalui lapisan *dense* dan fungsi aktivasi seperti *sigmoid* untuk menghasilkan nilai probabilitas kesamaan $p \in [0,1]$. Nilai ini kemudian dapat dinormalisasi menjadi skala interpretatif, agar lebih mudah dipahami sebagai tingkat kemiripan antar teks. Pendekatan probabilistik ini umum digunakan pada model *Semantic Textual Similarity* (STS) modern yang menggunakan pendekatan *Cross-Encoder*. Dengan demikian, model tidak hanya mengandalkan jarak antarvektor, tetapi juga belajar menyesuaikan fungsi kemiripan secara langsung dari data pelatihan melalui optimisasi berdasarkan nilai loss [66].

2.5 *BERT*

BERT (*Bidirectional Encoder Representations from Transformers*) adalah model representasi bahasa yang sudah dilatih (*pretrained*) berbasis *Transformer encoder* yang mempelajari konteks dua arah, lalu umumnya dilakukan proses *fine-tune atau* penyesuaian model dengan tugas spesifik dengan diikuti *layer* atau lapisan klasifikasi sederhana untuk beragam tugas pemahaman bahasa. Pendekatan ini menghasilkan lompatan kinerja pada berbagai *benchmark* tanpa arsitektur spesifik yang rumit [67].

Secara arsitektural, BERT dibangun sepenuhnya dari lapisan *Transformer encoder* yang bekerja berdasarkan mekanisme *self-attention*. Mekanisme ini memungkinkan setiap

token dalam kalimat memperhatikan seluruh token lain secara bersamaan, baik yang berada sebelum maupun sesudahnya [67]. Dengan cara ini, BERT dapat memahami hubungan dua arah (*bidirectional context*) antar kata di dalam satu kesatuan makna. Setiap lapisan *encoder* BERT terdiri atas komponen *multi-head self-attention*, *feed-forward network*, dan *layer normalization* yang berfungsi memperkaya representasi semantik pada setiap token. Proses berlapis ini menghasilkan vektor *embedding* kontekstual yang merepresentasikan makna kata sesuai dengan konteks kalimat secara menyeluruh [68].

Selain digunakan melalui proses *fine-tuning*, BERT juga dapat dimanfaatkan sebagai penyedia representasi teks atau *embedding* yang telah terlatih sebelumnya (*pretrained feature extractor*). Pada mode ini, lapisan-lapisan BERT tidak lagi diperbarui selama pelatihan, sehingga model berfungsi untuk menghasilkan representasi kontekstual yang tetap / statis (*frozen embeddings*). Pendekatan ini banyak digunakan pada penelitian yang ingin memanfaatkan kemampuan representasi semantik BERT tanpa memerlukan sumber daya komputasi besar yang dibutuhkan pada proses *fine-tuning* penuh. Meskipun bersifat statis terhadap parameter model, *embedding* yang dihasilkan tetap mengandung pemahaman konteks dua arah karena berasal dari arsitektur *Transformer encoder* yang telah dilatih pada korpus berskala besar [69].

Embedding yang dihasilkan oleh BERT dapat digunakan sebagai representasi awal dalam berbagai arsitektur jaringan saraf untuk tugas pemahaman bahasa. Representasi kontekstual tersebut umumnya menjadi masukan bagi lapisan pemrosesan lanjutan jaringan *feed-forward* yang berfungsi menafsirkan hubungan semantik antar token. Melalui pendekatan ini, model dapat memanfaatkan kekayaan makna yang terkandung dalam *embedding* BERT sekaligus memperkuat pemahaman terhadap struktur dan urutan kata dalam kalimat [70].

Dalam berbagai tugas yang melibatkan dua potongan teks, seperti *Semantic Textual Similarity (STS)*, terdapat dua pendekatan umum untuk memanfaatkan BERT, yaitu *Cross-Encoder* dan *Bi-Encoder* [64]. Pada pendekatan *Cross-Encoder*, kedua kalimat digabungkan ke dalam satu urutan *input* dengan format [CLS] Kalimat 1 [SEP] Kalimat 2 [SEP] [67]. Seluruh token dari kedua kalimat tersebut kemudian diproses secara bersamaan melalui lapisan *Transformer encoder*, sehingga setiap token dapat saling berinteraksi dan membentuk representasi yang mempertimbangkan konteks antar kalimat. Pendekatan ini mampu menangkap hubungan semantik yang lebih kaya, karena mekanisme *self-attention* beroperasi lintas kedua teks [71].

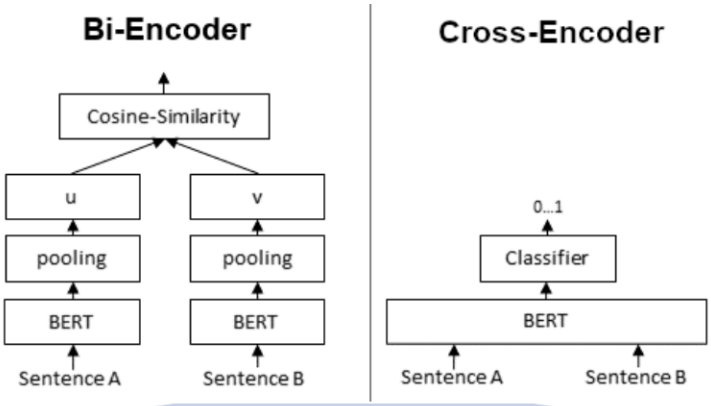
Sementara itu, pada pendekatan *Bi-Encoder*, setiap kalimat diproses secara terpisah menggunakan dua unit *encoder* yang identik untuk menghasilkan dua vektor representasi

independen. Nilai kesamaan kemudian dihitung menggunakan ukuran jarak seperti *cosine similarity*. Meskipun lebih efisien secara komputasi, pendekatan *Bi-Encoder* cenderung kehilangan detail interaksi langsung antar token dari dua kalimat. Oleh karena itu, *Cross-Encoder* lebih banyak digunakan pada penelitian yang menekankan akurasi pemahaman semantik, sedangkan *Bi-Encoder* lebih sering dipilih pada sistem yang memerlukan kecepatan tinggi dalam pencarian atau pemrosesan skala besar [71].

Sebelum diproses oleh BERT, teks terlebih dahulu melalui tahap *tokenisasi* untuk mengubah setiap kalimat menjadi potongan-potongan kata atau sub-kata (*tokens*) yang dapat dikenali oleh model. BERT menggunakan metode *WordPiece tokenizer* yang memecah kata menjadi unit terkecil berdasarkan frekuensi kemunculan dalam korpus pelatihan. Token khusus seperti [CLS] ditambahkan di awal urutan sebagai representasi keseluruhan kalimat, sedangkan [SEP] berfungsi sebagai pemisah antar kalimat pada tugas yang melibatkan dua teks. Setiap token kemudian dikonversi menjadi tiga jenis embedding, yaitu *token embedding*, *segment embedding*, dan *positional embedding*, yang digabungkan untuk membentuk representasi vektor awal sebelum diproses oleh lapisan *Transformer encoder* [67].

Melalui proses layer berlapis, BERT menghasilkan representasi kontekstual untuk setiap token yang telah mempertimbangkan hubungan antar kata di seluruh kalimat. Hasil keluaran model ini dapat diambil dalam beberapa bentuk tergantung kebutuhan tugas, seperti vektor [CLS] yang mewakili makna keseluruhan urutan untuk klasifikasi, representasi tiap token untuk tugas *sequence labeling*, atau rata-rata seluruh vektor token (*mean pooling*) untuk mendapatkan representasi kalimat secara umum. Representasi-representasi ini kemudian dapat digunakan sebagai fitur masukan bagi model lanjutan atau lapisan klasifikasi pada berbagai aplikasi pemahaman bahasa [67].

Meskipun setiap lapisan *encoder* pada BERT telah memiliki mekanisme *Layer Normalization* internal untuk menjaga stabilitas aktivasi, penambahan lapisan normalisasi tambahan setelah keluaran BERT sering diterapkan dalam berbagai arsitektur lanjutan. Normalisasi tambahan ini berfungsi untuk menyesuaikan skala dan distribusi *embedding* hasil BERT dengan lapisan pemrosesan berikutnya, terutama ketika model digunakan sebagai *feature extractor* yang tidak dilatih ulang (*frozen*). Dengan cara ini, nilai representasi yang dihasilkan menjadi lebih seragam dan stabil saat diteruskan ke jaringan lanjutan. Selain itu, penambahan *Layer Normalization* setelah BERT juga dapat membantu mempercepat proses konvergensi, menjaga kestabilan gradien, serta meningkatkan kemampuan generalisasi model pada data yang bervariasi [72].



Gambar 2.4 Model *Bi-Encoder vs. Cross-Encoder* [73]

Pada Layer *Normalization* dilakukan dengan menormalkan vektor masukan berdasarkan nilai rata-rata (*mean*) dan varians yang dihitung di sepanjang dimensi fitur. Persamaan matematisnya dapat dituliskan sebagai berikut [74]:

1. Perhitungan nilai rata-rata (*mean*)

$$\mu_i = \frac{1}{m}\sum_{k \in S_i} x_k \quad \text{.....(14)}$$

2. Perhitungan varians dan penambahan *epsilon*

$$var = \frac{1}{m}\sum_{k \in S_i}(x_k - \mu_i)^2 \quad \text{.....(15)}$$

$$var_{epsilon} = var + \varepsilon \quad \text{.....(16)}$$

3. Perhitungan standar deviasi

$$\sigma_i = \sqrt{var_{epsilon}} \quad \text{.....(17)}$$

4. Normalisasi nilai fitur

$$\hat{x}_i = \frac{1}{\sigma_i}(x_i - \mu_i) \quad \text{.....(18)}$$

2.6 *Bidirectional Long Short-Term Memory* (BiLSTM)

Long Short-Term Memory (LSTM) merupakan pengembangan dari *Recurrent Neural Network* (RNN) yang dirancang untuk mengatasi permasalahan *vanishing gradient* melalui penambahan mekanisme pengendali memori internal. Arsitektur LSTM tidak hanya menyimpan *hidden state*, tetapi juga mempertahankan *cell state* yang berperan sebagai jalur utama penyimpanan informasi jangka panjang. Melalui *cell state*, informasi penting dapat mengalir sepanjang urutan tanpa mengalami banyak perubahan, sehingga memungkinkan jaringan untuk mengingat konteks dari langkah sebelumnya dalam jangka waktu yang lebih panjang [75].

Inti dari LSTM terletak pada tiga gerbang utama, yaitu *forget gate*, *input gate*, dan *output gate* [75]. Setiap gerbang dikendalikan oleh fungsi aktivasi *sigmoid* (σ) yang

menghasilkan nilai antara 0 dan 1, yang menandakan seberapa besar informasi yang harus diteruskan atau diabaikan. Pertama, *forget gate* menentukan bagian dari memori sebelumnya (*cell state* terdahulu) yang perlu dilupakan. Nilai mendekati 0 berarti informasi dihapus, sedangkan nilai mendekati 1 berarti informasi dipertahankan. Selanjutnya, *input gate* mengatur informasi baru yang akan ditambahkan ke memori. Gerbang ini bekerja bersama *candidate cell state* yang menghasilkan kandidat informasi baru melalui fungsi aktivasi tanh untuk menambah atau memperbarui memori lama. Kombinasi kedua komponen ini menghasilkan *cell state* baru yang mencerminkan keseimbangan antara informasi lama dan informasi baru [76].

Setelah pembaruan memori dilakukan, *output gate* menentukan bagian mana dari memori yang akan dijadikan keluaran pada langkah waktu tersebut. Nilai keluaran ini disebut *hidden state* dan berfungsi sebagai representasi konteks yang dikirim ke lapisan berikutnya atau ke langkah waktu selanjutnya. Dengan adanya sistem gerbang yang saling berkoordinasi, LSTM dapat secara adaptif memutuskan kapan harus mengingat, memperbarui, atau melupakan informasi tertentu. Mekanisme ini menjadikan LSTM jauh lebih efektif dalam mempelajari hubungan jangka panjang pada data berurutan dibandingkan RNN konvensional yang hanya mengandalkan propagasi sederhana antar langkah waktu [76].

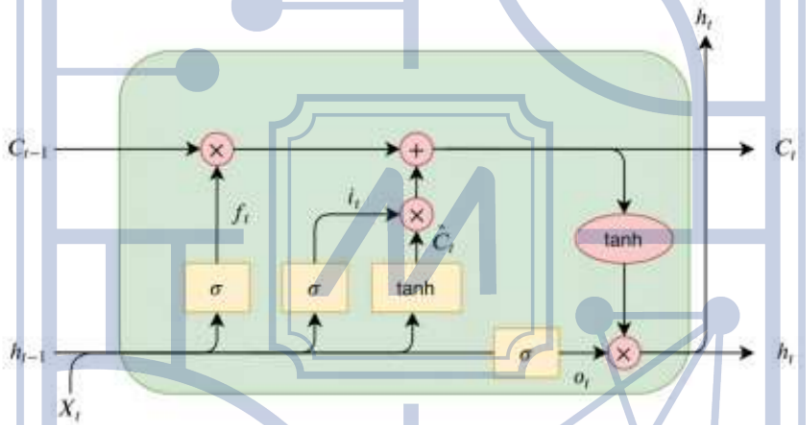
Selain itu, penggunaan tiga gerbang pengendali membuat LSTM lebih stabil terhadap masalah *vanishing* maupun *exploding gradient* selama proses pelatihan, sehingga model dapat belajar pola urutan yang kompleks tanpa kehilangan stabilitas numerik. LSTM juga memiliki kemampuan untuk membedakan antara konteks jangka pendek dan jangka panjang, sehingga informasi yang relevan dapat disimpan dan digunakan kembali sesuai kebutuhan pada setiap langkah waktu [75].

Karakteristik ini menjadikan LSTM unggul dalam berbagai tugas berbasis urutan seperti *machine translation*, *speech recognition*, *sentiment analysis*, dan *semantic textual similarity*. Dalam pemrosesan teks, kemampuan LSTM dalam memahami hubungan antar kata dan mempertahankan konteks sepanjang kalimat sangat penting untuk menghasilkan representasi makna yang lebih akurat dan kontekstual [76].

Salah satu pengembangan dari LSTM yang banyak digunakan dalam pemrosesan bahasa adalah *Bidirectional* LSTM (BiLSTM). Berbeda dengan LSTM standar yang hanya memproses urutan data dalam satu arah, BiLSTM memiliki dua lapisan LSTM yang berjalan secara paralel yaitu satu dari arah maju (*forward layer*) dan satu dari arah mundur (*backward layer*). Dengan cara ini, BiLSTM dapat menangkap informasi konteks baik dari sisi kiri maupun kanan dalam sebuah urutan teks [76].

Dalam pemrosesan teks, arah maju memungkinkan model memahami hubungan antar kata berdasarkan urutan alami kalimat, sedangkan arah mundur membantu model menangkap makna dari kata-kata yang muncul setelahnya. Kombinasi kedua arah ini menghasilkan representasi yang lebih kaya dan kontekstual, karena setiap kata direpresentasikan dengan mempertimbangkan seluruh konteks kalimat secara dua arah [77].

Keunggulan utama BiLSTM terletak pada kemampuannya memahami makna kata secara lebih menyeluruh dalam kalimat yang kompleks, termasuk ketika urutan atau posisi kata berpengaruh terhadap makna keseluruhan. Oleh karena itu, BiLSTM banyak diterapkan dalam berbagai tugas pemahaman teks seperti *named entity recognition*, *part-of-speech tagging*, *sentiment analysis*, dan *semantic textual similarity*, di mana konteks dua arah berperan penting dalam menentukan kesamaan atau makna semantik antarteks [78].



Gambar 2.5 Arsitektur LSTM [79]

Langkah-langkah inti dalam satu set LSTM berupa [80]:

1. *Forget Gate*

$$f_t = \sigma\left(W_f x_t + U_f h_{\{t-1\}} + b_f\right) \quad (19)$$

2. *Input Gate*

$$i_t = \sigma\left(W_i x_t + U_i h_{\{t-1\}} + b_i\right) \quad (20)$$

3. *Candidate Gate*

$$c_t = \tanh\left(W_g x_t + U_g h_{\{t-1\}} + b_g\right) \quad (21)$$

4. *Output Gate*

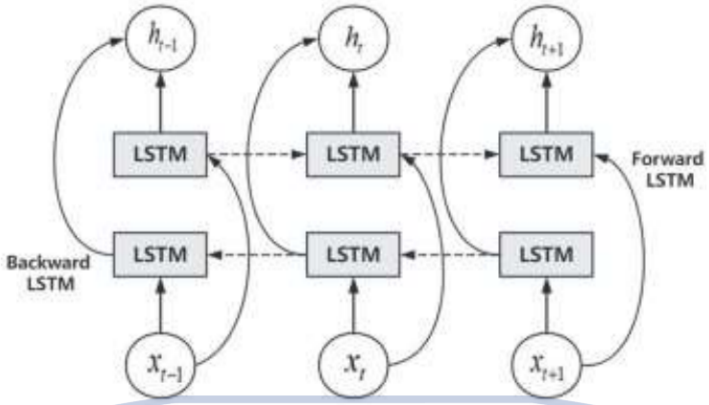
$$o_t = \sigma\left(W_o x_t + U_o h_{\{t-1\}} + b_o\right) \quad (22)$$

5. Pembaruan *Cell State*

$$c_t = f_t \odot c_{\{t-1\}} + i_t \odot c_t \quad (23)$$

6. Pembaruan *Hidden State*

$$h_t = o_t \odot \tanh(c_t) \quad (24)$$



Gambar 2.6 Model Arsitektur BiLSTM [81]

Secara sistematis proses perhitungan BiLSTM dalam memproses data dari dua arah dan menggabungkan hasil dapat ditulis sebagai berikut [82]:

1. Lapisan *Forward*

$$\overrightarrow{h_t} = \text{LSTM}^{\rightarrow}\left(x_t, \overrightarrow{h_{t-1}}\right) \quad (25)$$

2. Lapisan *Backward*

$$\overleftarrow{h_t} = \text{LSTM}^{\leftarrow}\left(x_t, \overleftarrow{h_{t-1}}\right) \quad (26)$$

3. *Output* Akhir

$$h_t = \left[\overrightarrow{h_t} \oplus \overleftarrow{h_t}\right] \quad (27)$$

2.7 *Attention*

Mekanisme *attention* merupakan salah satu konsep penting dalam pemrosesan bahasa alami modern. Konsep dasarnya terinspirasi dari cara manusia memusatkan perhatian pada bagian informasi yang dianggap paling relevan ketika memahami suatu konteks. Prinsip ini diterapkan dengan memberikan tingkat kepentingan yang berbeda pada setiap elemen dalam suatu representasi teks, sehingga bagian yang lebih relevan terhadap konteks atau tujuan tertentu dapat lebih ditonjolkan. Dengan demikian, mekanisme *attention* memungkinkan suatu sistem untuk memfokuskan pemrosesan pada bagian teks yang paling bermakna terhadap tugas yang sedang dijalankan, seperti memahami makna kalimat, menerjemahkan teks, maupun membandingkan kesamaan semantik antarteks [83].

Dalam perkembangannya, terdapat beberapa jenis *attention* yang dikembangkan dengan tujuan dan mekanisme perhitungan yang berbeda. Secara umum, perbedaan tersebut terletak pada cara model menghitung skor perhatian dan menentukan bagian *input* mana yang dianggap relevan. Beberapa varian yang paling dikenal antara lain *additive attention*, *multi-*

head attention, dan *self-attention* [68]. Masing-masing pendekatan memiliki karakteristik tersendiri dalam menentukan hubungan antar-token, di mana *additive attention* umumnya digunakan pada model berurutan seperti LSTM atau BiLSTM *[84]*, sedangkan *self-attention* banyak diadopsi pada model berbasis *Transformer* [68].

Additive attention, dikembangkan untuk memungkinkan model mengenali bagian *input* yang paling informatif tanpa bergantung pada perbandingan eksplisit antar-token seperti pada mekanisme *self-attention*. Pendekatan ini bekerja dengan mempelajari hubungan langsung antara keluaran tiap token pada lapisan sebelumnya dan parameter internal yang digunakan untuk menilai tingkat relevansi setiap token terhadap konteks keseluruhan. Melalui proses pembelajaran tersebut, model secara otomatis menyesuaikan fokusnya terhadap kata-kata yang memiliki makna paling relevan terhadap konteks kalimat. Hasil akhirnya berupa representasi terpusat (*context vector*) yang menonjolkan informasi penting dan menyederhanakan pemahaman semantik kalimat secara keseluruhan [68].

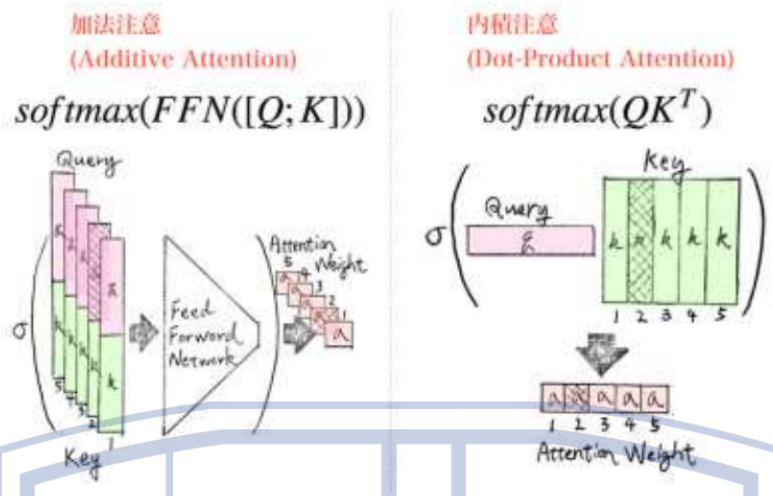
Mekanisme *additive attention* banyak digunakan sebagai pelengkap BiLSTM karena kemampuannya dalam merangkum informasi penting dari seluruh urutan keluaran BiLSTM menjadi satu representasi yang lebih terfokus. Dalam pemodelan berurutan, BiLSTM telah membentuk representasi kontekstual untuk setiap token, sehingga mekanisme *self-attention* tambahan sering kali tidak diperlukan. *Additive attention* berperan menyeleksi dan menekankan token-token yang paling relevan terhadap makna keseluruhan kalimat, sehingga menghasilkan vektor ringkasan yang representatif dan sesuai untuk tugas seperti *Semantic Textual Similarity* yang menitikberatkan perbandingan makna secara global.

Additive Attention bekerja dengan menilai tingkat relevansi setiap keluaran dari lapisan sebelumnya terhadap konteks keseluruhan yang sedang dipelajari. Proses ini dimulai dengan menghitung skor perhatian (*attention score*) untuk setiap representasi token, yang merepresentasikan seberapa besar kontribusinya terhadap makna keseluruhan kalimat. Skor tersebut kemudian dinormalisasi menggunakan fungsi *softmax* untuk menghasilkan bobot perhatian yang merepresentasikan besarnya kontribusi masing-masing token terhadap makna kalimat. Bobot ini digunakan untuk menggabungkan seluruh representasi token dengan menekankan bagian teks yang paling relevan, sehingga terbentuk *context vector* yang mewakili informasi utama dari kalimat. Vektor konteks tersebut menjadi ringkasan informasi yang menyoroti bagian teks paling relevan, membantu model memusatkan perhatian pada kata-kata bermakna dan mengabaikan informasi yang kurang penting [68].

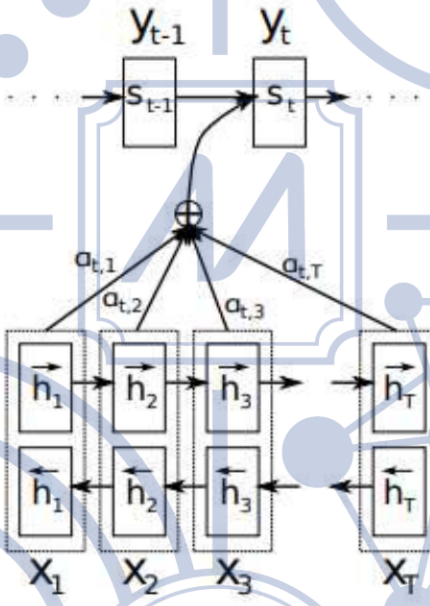
Self-attention merupakan mekanisme perhatian yang memungkinkan setiap token dalam suatu sekuens untuk secara langsung memodelkan hubungannya dengan token-token

lain dalam sekuens yang sama. Berbeda dengan *additive attention* yang menilai relevansi token terhadap suatu representasi global atau parameter internal, *self-attention* menghitung tingkat keterkaitan antartoken secara eksplisit dengan membandingkan representasi token menggunakan skema *query, key,* dan *value*. Pendekatan ini memungkinkan model untuk menangkap dependensi jarak dekat maupun jarak jauh secara efisien tanpa bergantung pada pemrosesan berurutan, sehingga sangat sesuai untuk pemodelan konteks global dalam teks [68].

Self-attention merupakan mekanisme perhatian yang memungkinkan setiap token dalam suatu sekuens untuk secara langsung memodelkan hubungannya dengan token-token lain dalam sekuens yang sama. Berbeda dengan *additive attention* yang menilai relevansi token terhadap suatu representasi global atau parameter internal, *self-attention* menghitung tingkat keterkaitan antartoken secara eksplisit dengan membandingkan representasi token menggunakan skema *query, key,* dan *value*. Pendekatan ini memungkinkan model untuk menangkap dependensi jarak dekat maupun jarak jauh secara efisien tanpa bergantung pada pemrosesan berurutan, sehingga sangat sesuai untuk pemodelan konteks global dalam teks [68]. Pendekatan *self-attention* menjadi komponen utama dalam arsitektur *Transformer* dan model berbasis *Transformer* seperti BERT, di mana seluruh token dalam satu sekuens diproses secara paralel. Melalui mekanisme ini, representasi setiap token diperkaya oleh informasi kontekstual dari token lain, sehingga *embedding* yang dihasilkan bersifat kontekstual dan dinamis. Hal ini memungkinkan model memahami hubungan semantik yang kompleks, termasuk relasi antarkata, frasa, dan struktur kalimat, secara lebih menyeluruh dibandingkan pendekatan berbasis pemrosesan sekuensial murni [67]. Pendekatan *self-attention* menjadi komponen utama dalam arsitektur *Transformer* dan model berbasis *Transformer* seperti BERT, di mana seluruh token dalam satu sekuens diproses secara paralel. Melalui mekanisme ini, representasi setiap token diperkaya oleh informasi kontekstual dari token lain, sehingga *embedding* yang dihasilkan bersifat kontekstual dan dinamis. Hal ini memungkinkan model memahami hubungan semantik yang kompleks, termasuk relasi antar kata, frasa, dan struktur kalimat, secara lebih menyeluruh dibandingkan pendekatan berbasis pemrosesan sekuensial murni [67].



Gambar 2.7 Perbedaan Mekanisme *Additive Attention* dan *Self-Attention* [85]



Gambar 2.8 Arsitektur Mekanisme *Additive Attention* [86]

Persamaan yang digunakan untuk proses *training additive attention* sebagai berikut [87]:

1. Proyeksi *attention*

$$a_t = W_h h_t + b_h \quad \text{.....(28)}$$

2. Aktivasi nonlinier

$$z_t = tanh(a_t) \quad \text{.....(29)}$$

3. Skor relevansi

$$e_t = v^T z_t \quad \text{.....(30)}$$

4. Bobot *attention*

$$\alpha_t = \frac{\exp(e_t)}{\sum_j \exp(e_j)} \quad (31)$$

5. Vektor konteks

$$s = \sum_t^T \alpha_t h_t \quad (32)$$

2.8 Evaluasi Model

Evaluasi model merupakan tahap penting dalam proses pembelajaran mesin yang bertujuan untuk menilai sejauh mana model mampu melakukan generalisasi terhadap data baru yang belum pernah dilihat sebelumnya. Proses ini tidak hanya memastikan bahwa model memiliki kinerja yang baik pada data pelatihan, tetapi juga mengukur ketepatannya dalam memprediksi pola pada data uji (*testing data*). Dengan demikian, evaluasi menjadi indikator utama dalam menentukan keberhasilan pelatihan dan efektivitas arsitektur model yang digunakan [88].

Pemantauan kinerja model selama proses pelatihan biasanya dilakukan pada setiap epoch dengan menggunakan beberapa metrik utama, yaitu *loss*, *accuracy* dan AUC (*Area Under the Curve*) [89]. *Loss* digunakan untuk mengukur besarnya kesalahan prediksi model terhadap nilai target berdasarkan fungsi kerugian yang digunakan sebagaimana tertera pada Persamaan (6), yang merepresentasikan perbedaan antara probabilitas prediksi model dan label aktual pada tugas klasifikasi biner. *Accuracy* merangkum proporsi total prediksi yang benar, baik kelas positif maupun negatif, terhadap keseluruhan data [88]. Metrik ini memberikan gambaran umum mengenai kinerja model secara keseluruhan. Sementara itu, AUC digunakan untuk mengukur kemampuan model dalam membedakan antara kelas positif dan negatif pada berbagai ambang batas probabilitas. Nilai AUC yang mendekati 1.0 menunjukkan daya pemisah yang sangat baik, sedangkan nilai mendekati 0.5 menunjukkan performa yang setara dengan tebakan acak [89].

Setelah pelatihan selesai, evaluasi performa model dilakukan secara lebih mendalam menggunakan metrik yang didasarkan pada *Confusion Matrix* yaitu alat yang digunakan untuk mengevaluasi kinerja model dengan membandingkan hasil prediksi dengan label sebenarnya. Matriks ini memvisualisasikan empat kemungkinan hasil prediksi [90]:

1. *True Positive* (TP): Jumlah observasi positif yang diklasifikasikan dengan benar.
2. *True Negative* (TN): Jumlah observasi negatif yang diklasifikasikan dengan benar.
3. *False Positive* (FP): Jumlah observasi negatif yang salah diklasifikasikan sebagai positif.
4. *False Negative* (FN): Jumlah observasi positif yang salah diklasifikasikan sebagai negatif.

Dari keempat komponen ini, berbagai metrik turunan dapat dihitung untuk memberikan gambaran yang lebih terperinci mengenai kemampuan model dalam mengenali kelas positif maupun negatif. Salah satu metrik yang paling umum digunakan adalah *Precision*, yang mengukur ketepatan prediksi positif, yakni seberapa besar proporsi data yang diprediksi positif benar-benar termasuk ke dalam kelas positif. Selanjutnya, *Recall* menilai sejauh mana model mampu menemukan seluruh data positif yang sebenarnya ada, sedangkan *F1-Score* digunakan untuk menyeimbangkan kedua metrik tersebut sehingga dapat menggambarkan kinerja model secara lebih menyeluruh.

Secara matematis, metrik evaluasi yang digunakan dalam penelitian ini dirumuskan sebagai berikut [90]:

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \quad (33)$$

$$AUC = \frac{1}{PN \sum_{i=1}^{m-1} (FP_{i+1} - FP_i) \times \frac{TP_{i+1} + TP_i}{2}} \quad (34)$$

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \quad (35)$$

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \quad (36)$$

$$F_{1\text{-Score}} = 2 \times \frac{\mathrm{Presisi} \times \mathrm{Recall}}{\mathrm{Presisi} + \mathrm{Recall}} \quad (37)$$