BAB II
KAJIAN LITERATUR

2.1 *Natural* Language Processing

Pemrosesan bahasa alami (*Natural Language Processing*) merupakan salah satu cabang kecerdasan buatan (*Artificial Intelligence*) yang berfokus pada pengembangan metode agar komputer mampu berinteraksi dengan manusia menggunakan bahasa alami. Tujuan utama NLP adalah memungkinkan mesin untuk memahami, menafsirkan, menganalisis, serta menghasilkan bahasa manusia secara bermakna sehingga dapat digunakan dalam berbagai aplikasi berbasis teks maupun suara. Kompleksitas bahasa alami, seperti adanya ambiguitas makna, variasi struktur kalimat, serta perbedaan konteks, menjadikan NLP sebagai bidang penelitian yang menantang sekaligus memiliki peluang besar dalam pengembangan teknologi cerdas. Oleh karena itu, NLP terus berkembang untuk menghasilkan sistem yang mampu memproses bahasa secara lebih akurat dan menyerupai kemampuan manusia dalam memahami komunikasi sehari-hari [15].

Perkembangan NLP mengalami perubahan yang cukup signifikan dari waktu ke waktu. Pada tahap awal, sebagian besar sistem NLP dibangun menggunakan pendekatan berbasis aturan (*rule-based approach*) yang mengandalkan aturan tata bahasa dan pengetahuan linguistik yang disusun secara manual oleh pakar. Meskipun pendekatan ini mampu memberikan hasil yang baik pada domain tertentu, sistem yang dihasilkan cenderung sulit dikembangkan, kurang fleksibel terhadap variasi bahasa, serta membutuhkan biaya pemeliharaan yang tinggi ketika dihadapkan pada data dalam skala besar. Keterbatasan tersebut mendorong berkembangnya pendekatan berbasis data (*data-driven approach*) yang lebih adaptif terhadap perubahan pola bahasa.

Seiring berkembangnya pendekatan berbasis data, metode statistik mulai banyak diterapkan dalam NLP melalui algoritma seperti *Hidden Markov Model* (HMM) dan *Conditional Random Fields* (CRF). Pendekatan ini memungkinkan pemodelan hubungan antar kata dan struktur kalimat secara probabilistik sehingga mampu meningkatkan kinerja berbagai tugas NLP dibandingkan metode berbasis aturan. Selanjutnya, perkembangan *machine learning* membawa perubahan yang lebih besar dengan memungkinkan model mempelajari karakteristik bahasa secara langsung dari data tanpa harus mendefinisikan aturan secara eksplisit. Berbagai algoritma *supervised learning* maupun *unsupervised learning* kemudian banyak diterapkan untuk tugas-tugas seperti klasifikasi teks, analisis sentimen, pengelompokan dokumen, serta penerjemahan mesin.

Pada awalnya, NLP menggunakan pendekatan berbasis aturan (*rule-based*) yang bergantung pada kaidah linguistik, namun metode ini memiliki keterbatasan dalam hal fleksibilitas dan skalabilitas. Seiring meningkatnya kemampuan komputasi, pendekatan statistik mulai dikembangkan melalui teknik *Hidden Markov Models* (HMM) dan *Conditional Random Fields* (CRF), yang memungkinkan pemodelan bahasa secara probabilistik dan menandai pergeseran ke pendekatan berbasis data. Selanjutnya, penerapan machine learning membawa perubahan besar dengan memungkinkan sistem mempelajari pola langsung dari data. Metode supervised learning dan unsupervised learning banyak digunakan dalam berbagai tugas NLP, seperti analisis sentimen, klasifikasi teks, dan penerjemahan mesin, dengan memanfaatkan dataset dalam jumlah besar. Dalam beberapa tahun terakhir, deep learning semakin mendominasi NLP. Arsitektur seperti *Recurrent Neural Networks* (RNN), *Long Short-Term Memory* (LSTM), dan *Convolutional Neural Networks* (CNN) terbukti efektif dalam meningkatkan performa berbagai tugas pemrosesan bahasa. Perkembangan semakin pesat dengan hadirnya model transformer yang menggunakan mekanisme attention, seperti BERT, GPT, dan *Text-To-Text Transfer Transformer* (T5) yang mampu memahami konteks bahasa secara lebih baik dan mencapai kinerja unggul pada berbagai aplikasi [16].

2.2 Automatic Question Generation (AQG)

Automatic Question Generation (AQG) merupakan suatu sistem yang secara otomatis menghasilkan pertanyaan berdasarkan topik, gagasan, atau konteks tertentu. Pertanyaan tersebut dibuat dari informasi yang terdapat dalam teks paragraf maupun gambar, kemudian diolah menggunakan bahasa alami sehingga membentuk pertanyaan yang relevan dan sesuai dengan isi sumbernya [17]. AQG menjadi salah satu pemanfaatan teknologi yang strategis dalam bidang pendidikan. Sistem AQG memanfaatkan *Natural Language Processing* (NLP) serta pembelajaran mesin *Machine Learning* (ML) untuk menghasilkan beragam pertanyaan secara otomatis dari konten tekstual [18]. Otomatisasi dalam pembuatan soal semakin penting seiring meningkatnya tren pembelajaran daring serta kebutuhan skalabilitasnya dalam beberapa tahun terakhir [19]. Tujuan akhir dari sistem AQG adalah supaya sistem mampu membuat pertanyaan yang susunan katanya benar, maknanya jelas, dan sesuai dengan konteks yang sedang dibahas. Jadi, pertanyaan yang dihasilkan tidak terasa aneh atau membingungkan, tetapi alami dan mudah dipahami seperti pertanyaan yang dibuat oleh manusia [17].

2.2.1 Manfaat *Automatic Question Generation*

Penerapan sistem pembuatan pertanyaan otomatis atau *Automatic Question Generation* (AQG) memberikan berbagai manfaat dalam pembelajaran, baik bagi pelajar maupun pengajar, yaitu:

1. Bagi pelajar, AQG dapat mendukung proses belajar mandiri dengan merekomendasikan pertanyaan yang berfokus pada konsep-konsep kunci dalam materi pembelajaran, sehingga membantu pelajar memahami isi materi secara lebih terarah [20].
2. Bagi pengajar, AQG dapat dimanfaatkan sebagai alat bantu dalam menyusun pertanyaan pembelajaran. Sistem ini tidak menggantikan peran pengajar sebagai pengambil keputusan akhir, melainkan membantu menghasilkan pertanyaan yang relevan, sesuai dengan topik, serta selaras dengan tingkat peserta didik. Oleh karena itu, AQG berpotensi mengurangi waktu dan usaha yang diperlukan dalam proses penyusunan soal [21].

2.2.2 Metode *Automatic Question Generation* (AQG)

Perkembangan metode *Automatic Question Generation* (AQG) telah mengalami pergeseran paradigma yang signifikan seiring dengan kemajuan teknologi pemrosesan bahasa alami. Secara umum, evolusi metode ini dapat diklasifikasikan ke dalam tiga fase pendekatan utama, yaitu:

1. *Rule-Based* (Berbasis Aturan)

Pada tahap awal pengembangannya, bidang AQG didominasi oleh pendekatan berbasis aturan (*rule-based approaches*). Metode ini mengandalkan aturan *linguistik manual* untuk mengubah kalimat input menjadi pertanyaan. Namun, seiring berjalannya waktu, bidang ini secara progresif beralih meninggalkan metode berbasis aturan menuju metode berbasis jaringan saraf (*neural network-based*) untuk mendapatkan hasil yang lebih baik [22].

2. *Sequence-to-Sequence* (Seq2Seq)

Sequence-to-Sequence (Seq2Seq) merupakan salah satu pendekatan deep learning yang banyak digunakan dalam *Automatic Question Generation* (AQG). Pendekatan ini menitikberatkan pada pemanfaatan arsitektur Seq2Seq untuk menghasilkan pertanyaan dari teks sumber. Kajian dalam pendekatan ini mencakup berbagai model jaringan saraf yang digunakan untuk tugas pembangkitan pertanyaan, termasuk metode yang mempertimbangkan keberadaan jawaban dalam proses generasi (*answer-aware*) maupun metode yang tidak bergantung pada informasi jawaban (*answer-agnostic*) [23].

3. *Transformer*

Fase terkini dalam perkembangan AQG didominasi oleh arsitektur *Transformer* yang memperkenalkan mekanisme *attention* untuk memproses data secara lebih efisien melalui *pre-training* model. Arsitektur ini kemudian berkembang menjadi *Large Language Models* (LLMs) yang memiliki miliaran parameter dan dilatih pada dataset yang sangat luas, sehingga mampu menangani berbagai tugas teks mulai dari generasi soal hingga sistem tutorial [23].

2.3 Preprocessing

Preprocessing adalah tahapan awal yang sangat krusial dalam proses pengolahan data, khususnya pada bidang *Natural Language Processing* (NLP). *Preprocessing* bertujuan untuk mempersiapkan data mentah agar memiliki kualitas yang baik, terstruktur, dan sesuai dengan kebutuhan model yang akan digunakan. Data teks yang diperoleh dari berbagai sumber umumnya bersifat tidak terstruktur, mengandung *noise*, serta memiliki format yang tidak konsisten, sehingga tidak dapat langsung digunakan dalam proses pelatihan model.

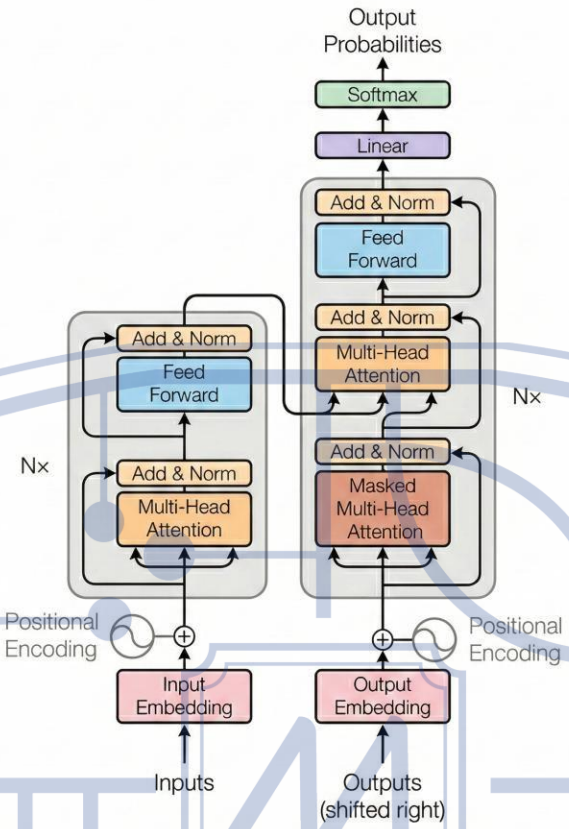
Preprocessing dianggap sebagai langkah fundamental dalam *pipeline* NLP karena secara langsung memengaruhi performa model. Proses seperti pembersihan data, normalisasi, dan tokenisasi terbukti mampu meningkatkan kualitas representasi teks sehingga model dapat menghasilkan output yang lebih akurat dan relevan [24]. Dalam penelitian ini, tahapan *preprocessing* disusun secara sistematis untuk menyesuaikan kebutuhan model *Automatic Question Generation* (AQG) berbasis *transformer* [25]. Adapun tahapan *preprocessing* yang dilakukan meliputi *undersampling*, pembersihan data, transformasi format, pembagian dataset, serta tokenisasi :

1. Teknik *Undersampling*, merupakan strategi pra-pemrosesan yang esensial untuk memitigasi bias model akibat ketidakkeseimbangan data melalui eliminasi sampel kelas mayoritas secara strategis. Pendekatan ini tidak hanya meningkatkan sensitivitas model terhadap karakteristik kelas minoritas yang sering kali merupakan informasi paling kritis dalam data pendidikan, tetapi juga secara signifikan mereduksi dimensi data sehingga menciptakan efisiensi komputasi dan manajemen memori yang jauh lebih optimal dibandingkan metode penyeimbangan data lainnya. Implementasi undersampling memberikan landasan yang kuat bagi algoritma klasifikasi untuk membentuk batas keputusan yang objektif, yang secara langsung mendukung performa sistem dalam memberikan evaluasi adaptif yang presisi dan responsive [26].

2. Pembersihan data ini dilakukan dengan proses penghilangan berbagai elemen yang tidak relevan seperti karakter khusus, simbol, URL, serta tanda baca yang berlebihan. Selain itu, dilakukan normalisasi teks dengan mengubah seluruh huruf menjadi huruf kecil (*lowercase*) untuk menjaga konsistensi data. Proses ini bertujuan untuk mengurangi *noise* dalam data sehingga informasi yang diproses oleh model menjadi lebih fokus dan bermakna [25].
3. Transformasi data merupakan tahapan krusial dalam praproses data yang bertujuan untuk mengubah data mentah dari satu struktur ke struktur lainnya sesuai dengan volume, kompleksitas, dan format yang dibutuhkan untuk analisis. Proses ini memastikan bahwa data berada dalam bentuk yang dapat diterima dan sesuai untuk model prediksi atau klasifikasi [27].
4. Pembagian dataset atau *data spliting* merupakan salah satu proses yang krusial untuk mendapatkan performa model yang andal dan konsisten [28]. Tujuan dari pembagian dataset ini adalah untuk menghindari *overfitting*. Pada pembagian dataset dapat dibagi menjadi tiga bagian yaitu data latih (*training set*) yang sebaiknya merupakan bagian terbesar, data validasi (*cross-validation set* atau *development set*) dan data uji (*testing set*) dimana data validasi dan data uji dapat memiliki proporsi yang sama. Dalam beberapa kasus ditemukan bahwa peneliti menghilangkan data uji dan hanya membagi dataset menjadi dua bagian yang biasanya disebut sebagai development set dan test set [29]. Salah satau proporsi pembagian data yang sering digunakan adalah 80% : 10% : 10%, dimana 80% untuk data training, 10% untuk data validation dan 10% untuk data testing [30].

2.4 *Transformer*

Transformer merupakan salah satu arsitektur jaringan saraf modern dalam pemrosesan bahasa alami atau NLP yang memiliki kemampuan kuat dalam menangkap hubungan kontekstual pada data berurutan melalui mekanisme perhatian (*attention*). Kemampuan ini membuat *Transformer* banyak digunakan tidak hanya pada tugas NLP, tetapi juga pada berbagai bidang lain seperti *transfer learning* dan deteksi anomali [31]. Berikut disajikan gambar terkait arsitektur *Transformer*



Gambar 2.1 *Transformer Architecture* [32].

Perkembangan lebih lanjut dari arsitektur ini melahirkan berbagai model turunan yang sangat berpengaruh, seperti BERT, GPT, dan T5, yang terbukti memberikan kontribusi signifikan terhadap kemajuan berbagai tugas NLP, termasuk klasifikasi teks, sistem tanya jawab (*question answering)*, dan penerjemahan mesin [4].

2.4.1 *Encoder*

Encoder terdiri dari 6 lapisan yang identik, dimana setiap lapisan memiliki dua sub-lapisan yaitu, pertama *multi-head self-attention mechanism* dan yang kedua adalah jaringan *feed-forward* yang sepenuhnya terhubung secara posisional. Di sekitar masing-masing dari kedua sub-lapisan digunakan koneksi residual dan diikuti oleh *Layer Normalization*. Untuk memahami lebih lanjut struktur pada *encoder*, berikut dijelaskan komponen – komponen utama yang menyusunnya [22]:

1. Tokenisasi

   Pada model transformer modern seperti T5, proses pemrosesan teks tidak lagi menggunakan tokenisasi berbasis kata utuh (*word-level*), melainkan berbasis subkata

(*subword-level*). Hal ini bertujuan untuk mengatasi masalah *Out-of-Vocabulary* (OOV) [34,35].

T5 secara spesifik menggunakan *SentencePiece* yang di dalamnya sering mengimplementasikan algoritma *Unigram Language Model*. Berbeda dengan BPE (*Byte Pair Encoding*), Unigram memulai dari *vocabulary* yang besar dan membuang token secara progresif. Probabilitas sebuah kalimat S yang didekomposisi menjadi urutan token $x = (x_i, \dots, x_n)$ dirumuskan sebagai:

$$P(x) = \prod_{i=1}^{n} p(x_i) \quad (1)$$

Di mana V adalah kumpulan semua kandidat subkata, dan algoritma mencari segmentasi yang memaksimalkan *likelihood*:

$$S^* = \arg\max_{x \in S(V)} P(X) \quad (2)$$

Untuk menyeleksi kanditat sub kata, dapat dilakukan melalui perhitungan *Marginal Likelihood* yang bertujuan untuk memaksimalakan probabilitas kemunculan seluruh teks dalam korpus dengan pilihan subkata yang ada di *vocabulary*. Beikut persamaan *Marginal Likelihood*:

$$\mathcal{L} = \sum_{s=1}^{M} \log \sum_{x \epsilon S(d_s)} P(x) \quad (3)$$

Di mana M adalah Total jumlah kalimat dalam korpus, d_s adalah kalimat ke-s dalam korpus, $S(d_s)$ adalah kumpulan semua kemungkinan cara memotong kalimat d_s menjadi subkata yang ada di V, dan $P(X)$ adalah probabilitas dari urutan subkata tersebut.

2. *Embeddings*

   Seperti pada model *sequence transduction* lainnya, arsitektur *transformer* menggunakan *embbedding* yang dipelajari (*learned embeddings*) untuk mengubah token masukkan dan keluaran kedalaman bentuk representasi vektor dengan dimensi d_{model}. Proses ini memungkinkan model untuk merepresentasikan kata dalam bentuk numerik yang dapat diproses untuk proses selanjutnya. Pada implementasinya, model menggunakan matriks bobot yang sama (*weight sharing*) antar lapisan *embedding input*, *embedding output* dan transformasi linear sebelum *softmax* [35].

3. *Attention*

   Mekanisme *self-attention* merupakan elemen inti dalam arsitektur *Transformer* yang diperkenalkan oleh Vaswani et al. (2017). Mekanisme ini memungkinkan setiap token dalam suatu urutan saling berinteraksi dengan token lainnya melalui perhitungan bobot perhatian yang menunjukkan tingkat keterkaitan antar token [32]. Fungsi *attention*

dapat diilustrasikan sebagai pemetaan kueri dan sekumpulan pasangan kunci – nilai terhadap suatu keluaran. Untuk memperolah *Query* (Q), *Key* (K) dan *Value* (V), input embedding terlebih dahulu ditransformasikan menggunakan proyeksi linear dengan matriks bobot yang dapat dipelajari, adapun rumus yang digunakan sebagai berikut [35].

$$Q = XW_Q, K = XW_K, V = XW_V \quad (4)$$

Dimana:

$Q = Query$

$K = Key$

$V = Value$

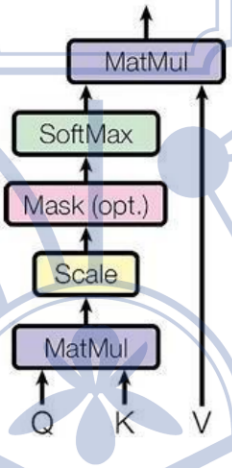
$W_Q = \text{Bobot Q}$

$W_K = \text{Bobot K}$

$W_V = \text{Bobot V}$

4. *Scaled Dot-Product Attention*

Inputan pada mekanisme *attention* disebut sebagai *Scaled Dot-Product Attention.*



Gambar 2.2 *Scaled Dot-Product Attention* [36].

Dalam gamber 2.2 *Scaled Dot-Product Attention,* inputan terdiri dari Q (*query*) K (*key*) dan V (*value*). Bobot value diperoleh dengan melakukan perkalian *dot* (*dot product*) antara *query* dengan semua *key*, kemudian membaginya dengan $\sqrt{d_k}$ lalu menerapkan *softmax* pada perhitungannya [35].

Secara matematis, mekanisme *Self-Attention* dirumuskan sebagai berikut:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \quad V \quad (5)$$

Penjelasan:

Q (*Query*) representasi token yang sedang mencari informasi,

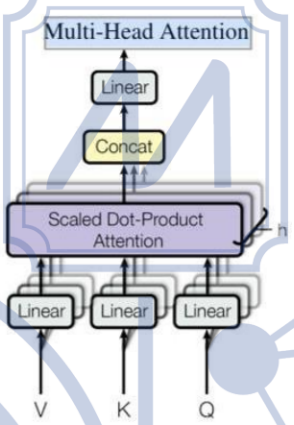
K (*Key*) representasi token pembanding,

V (*Value*) representasi token pembanding,

d_k dimensi vektor *key* yang digunakan untuk *scaling* agar nilai *dot-product* tidak terlalu besar [36].

5. *Multi-Head Attention*

Multi-head attention merupakan pengembangan dari mekanisme *attention* pada arsitektur *trasformer* yang bertujuan untuk meningkatkan kemampuan model dalam menangkap berbagai hubungan antar kata dalam satu waktu. Berbeda dengan *single-head attention* yang hanya menggunakan satu representasi, pendekatan ini melakukan beberapa proyeksi linear terhadap *query* (Q), *key* (K) dan *value* (V) kedalam beberapa sub-ruang representasi yang berbeda.



Gambar 2.3 *Multi Head Attention* [36].

Setiap hasil proyeksi tersebut kemudian diproses secara paralel melalui mekanisme *attention*, sehingga menghasilkan beberapa output yang disebut sebagai *head*. Seluruh output dari masing-masing *head* kemudian digabungkan (*concatenate*) dan diproyeksasikan kembali untuk menghasilkan representasi akhir.

Secara sistematis, *Multi-head attention* dapat dirumuskan sebagai berikut:

$$MultiHead(Q,K,V) = Concat(\text{head}_1,\dots,\text{head}_h)W^o \quad (6)$$

$$where\ \text{head}_i = attention(QW_i^Q, KW_i^K, VW_i^V) \quad (7)$$

Penjelasan rumus:

MultiHead (Q,K,V) = penggabungan 3 input $(Query, Key, Value)$

$Concat(\text{head}_1,\dots,\text{head}_h$ = penggabungan tiap hasil perhitungan kepala lapisan

w^O= Matriks proyeksi akhir

ead_i = kepala lapisan

$attention(QW_i^Q, KW_i^K, VW_i^V)$ = penerapan operasi perhatian pada *Quer*. *Key*, dan *Value* [35].

6. *Feed-Forward Networks*

Selain dari sub lapisan *attention*, setiap lapisan pada *encoder* dan *decoder* berisi jaringan *Feed-Forward* yang terhubung sepenuhnya. Jaringan ini diterapkan secara terpisah dan identik untuk setiap posisinya. FFN terdiri dari dua transformasi dengan aktivasi ReLu diantaranya.

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2 \quad (8)$$

Dimana :

W_1, W_2 = matiks bobot,

b_1, b_2 = bias

Meskipun transformasi linear yang digunakan sama untuk posisi yang berbeda, parameter yang digunakan untuk tiap lapisan akan berbeda [35].

7. *Positional Encoding*

Berbeda dengan model berbasis *Recurrent Neural Networks* maupun *Convolutional Neural Networks*, arsitektur *transformer* tidak memiliki mekanisme rekurensi maupun konvolusi. Oleh karena itu, diperlukan suatu metode untuk memberikan informasi terkait urutan posisi token dalam suatu kalimat. Untuk mengatasi itu, digunakan *positional encoding*, yaitu vektor tambahan yang digabungkan dengan *embedding input* pada bagian awal *encoder* dan *decoder*. Vektor ini memiliki dimensi yang sama dengan *embedding* yaitu d_{model}, sehingga keduanya dapat dijumlahkan secara langsung.

Terdapat berbagai pendekatan dalam menentukan *positional encoding*, baik yang dipelajari (*learned*) maupun yang bersifat tetap (*fixed*). Dalam pendekatan *Transformer*, digunakan fungsi sinusoidal berupa sinus dan cosinus dengan frekuensi yang berbeda untuk setiap dimensi vektor. Secara matematis, *positional encoding* dirumuskan sebagai berikut:

$$PE(pos, 2i) = \sin\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \quad (9)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \quad (10)$$

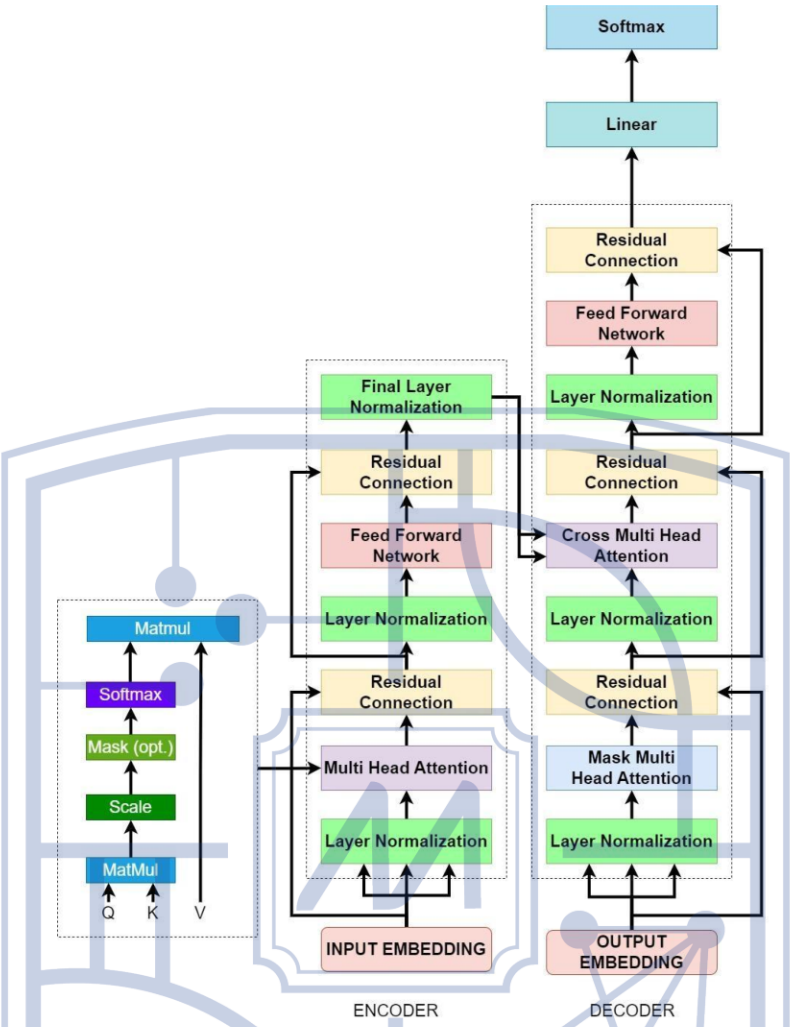
Dimana pos menunjukkan posisi token dalam urutan, dan i adalah indeks dimensi. Setiap dimensi pada *positional encoding* merepresentasikan gelombang sinusoidal dengan panjang gelombang yang berbeda, membentuk pola progresi geometris [35].

2.4.2 *Decoder*

Decoder terdiri dari 6 lapisan yang identik, dimana setiap lapisan *encoder* juga terdapat pada *decoder.* Setiap sub-lapisan pada *decoder* juga dilengkapi dengan koneksi residual dan diikuti oleh proses normalisasi lapisan (*Layer Normalization*) sama seperti pada *encoder*. Pada *decoder* terdapat penambahan satu sub-lapisan yang bertujuan untuk melakukan *multi-head attention* pada keluaran dari *encoder*. Selain itu, pada sub-lapisan *self attention* pada *decoder* dilakukan modifikasi berupa *masking* untuk mencegah setiap posisi memperhatikan token setelahnya. Mekanisme ini bertujuan untuk mempertahankan sifat *auto-regresif*, sehingga prediksi pada posisi ke-i hanya bergantung pada token sebelumnya. Pada bagian akhir *decoder,* layer terakhir akan melalui transformasi linear kemudian diikuti fungsi *softmax* yang digunakan untuk mengubah *output decoder* menjadi probabilitas token berikutnya yang diprediksi [22].

2.5 Text-to-Text-Transfer Transformer (T5)

Text-to-Text Transfer Transformer (T5) merupakan sebuah arsitektur *transformer* yang dikembangkan dengan mendefinisikan seluruh tugas *Natural Language Processing* (NLP) ke dalam format "teks-ke-teks" [37]. Berbeda dengan model pendahulunya seperti BERT yang hanya menggunakan *encoder* atau GPT yang hanya menggunakan *decoder*, T5 mengadopsi struktur *encoder-decoder* secara penuh sehingga mampu menangani berbagai variasi tugas NLP secara optimal [38]. Pendekatan ini membuat berbagai tugas bahasa bisa diubah ke format yang sama, yaitu berupa teks masuk dan teks keluar, sehingga lebih sederhana dan mudah digunakan [39]. Sehingga model pra-latih berbasis *transformer* seperti T5 bisa dimanfaatkan pada berbagai tugas NLP [40]. Karena fleksibilitasnya, T5 sering digunakan dalam penelitian pembangkitan teks, termasuk untuk membuat sistem yang dapat menghasilkan pertanyaan secara otomatis (AQG). Hal ini juga didukung oleh beberapa penelitian yang mengatakan bahwa model T5 dapat digunakan untuk tugas *Question Generation* (QG) dan *Question Answering* (QA) secara bersamaan [41].



Gambar 2.4 *Text-to-Text Transfer Transformer Architecture* [42].

Terdapat perbedaan pendekatan yang diambil oleh model T5 sebagai penyederhanaan dan memperbaiki beberapa aspek dari arsitektur *Transformer* yang asli yaitu :

1. *Positional Encoding*

   Pada *Transformer* urutan posisi token pada suatu kalimat digunakan *positional encoding* yaitu sebuah vektor tambahan yang digabungkan dengan *embedding input* pada bagian awal *encoder* dan *decoder* [35], sedangkan pada model T5 menggunakan bentuk *embedding* posisi yang lebih sederhana, dimana setiap *embedding* hanyalah sebuah skalar ditambahkan ke *logit* yang digunakan untuk menghitung bobot *attention* Sehingga rumus attention pada T5 dituliskan seperti berikut [42].

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + B\right) \quad V \tag{11}$$

   Dimana :

   QK^T : Skor Attention biasa

   B : Relative position bias

V : Matriks Value

2. *Feed Forward Netwok*

Pada transformer, pada *SubLayer Feed Forward Network* ditambahkan bias didalam rumusnya sedangkan pada T5 penggunaannya mengikuti standar dari Transofrmer akan tetapi penambahan bias tersebut dihilangkan. Sehingga rumusnya dapat dituliskan sebagai berikut:

$$FFN(x) = \max(0, xW_1)\,W_2 \quad (12)$$

3. *Layer Normalization*

Pada arsitektur *transformer*, *Layer Normalization* diakukan setelah proses *self attention*, *feef forward*, dan penambahan koneksi residual selesai. Sedangkan pada T5 *Layer Normalization* dilakukan dengan versi yang lebih sederhana, dimana nilai aktivasi hanya diskalakan dan tidak ada bias aditif yang diterapkan. *Layer Normalization* pada model T5 di lakukan sebelum masuk kedalam proses *self attention* dan *feedforward* [42].

4. *Residual Connection*

Pada Transformer, residual koneksi dilakukan setelah *SubLayer* selesai di lakukan. Dimana rumus *Residual Connection* dapat ditulis sebagai berikut [42]:

$$\textit{Residual Connection} = x + Sublayer(x) \quad (13)$$

Dimana:

x : *Input* ke *SubLayer*

SubLayer: *Output* dari *SubLayer*

Sedangkan pada T5, Terdapat perubahan dikarenakan perbedaan posisi *Layer Normalization*, sehingga *Residual Connection* pada T5 dapat dituliskan sebagai berikut:

$$\textit{Residual Connection} = x + Sublayer(LayerNorm(x)) \quad (14)$$

Dimana:

x : Input ke *SubLayer*

$SubLayer(LayerNorm)$: *Output* akhir dari *SubLayer* dimana yang sebelumnya melewati LayerNorm terlebih dahulu lalu masuk ke *SubLayer*.

5. Mekanismes *Dropout*

Residual *Dropout* merupakan teknik *regularisasi* yang diterapkan pada arsitektur *transformer* untuk mengurangi risiko *overfitting*. *Dropout* diterapkan pada keluaran setiap sub-layer sebelum hasil tersebut dijumlahkan dengan input sub-layer melalui

mekanisme residual dan kemudian dinormalisasi. Selain itu, *dropout* juga diterapkan pada hasil penjumlahan antara *embedding* dan *positional encoding*, baik pada bagian *encoder* maupun *decoder* [35]. Sedangkan Pada model T5 mekanisme *dropout* tidak hanya diterapkan pada bagian tersebut, tetapi juga digunakan didalam jaringan *feed-forwad* pada skip koneksi, pada bobot *attention* serta pada input dan output dari seluruh tumpukan [42].

2.5.1 T5-*Small*

T5-*Small* merupakan salah satu ukuran dari model T5 yang memiliki jumlah parameter lebih kecil dibandingkan *T5-Base*, dan *T5-Large* Dimana *T5-Small* memiliki jumlah parameter sebanyak 60 juta parameter, *T5-Base* memiliki jumlah parameter sebanyak 220 juta parameter dan *T5-Large* memiliki parameter sebanyak 770 juta parameter [42]. Perbedaan ukuran ini menunjukkan bahwa setiap varian model memiliki kapasitas parameter yang berbeda. Adanya variasi ukuran tersebut memungkinkan model T5 digunakan dalam berbagai konteks, sehingga dapat disesuaikan dengan ketersediaan sumber daya komputasi serta kebutuhan kinerja yang diinginkan [39].

2.5.2 Fungsi *Loss*

Dalam jaringan saraf tiruan, fungsi *loss* digunakan sebagai indikator matematis untuk mengukur seberapa jauh perbedaan antara prediksi yang dihasilkan oleh model dengan target kebenaran (*ground truth*). Pada tugas pembangkitan bahasa alami dan klasifikasi token seperti yang dilakukan arsitekur T5, fungsi yang digunakan adalah *Cross-Entropy Loss*.

Fungsi *Cross-Entropy* sangat efektif untuk mengoptimalkan tugas klasifikasi *multi-class*. Secara matematis, *Cross-Entropy Loss* (L_{CE}) dihitung menggunakan persamaan berikut [43]:

$$(L_{CE}) = \sum_{i=1}^{V} y_i \log(\hat{y}_i) \quad (15)$$

Keterangan:

V = ukuran *vocabulary size*

y_i = Nilai probabilitas target sebenarnya

$\hat{y}_i$ = Probabilitas prediksi keluaran model untuk kata ke-i

2.5.3 *Forward Pass*

Forward pass adalah fase eksekusi di mana aliran data masukan diproses melewati seluruh lapisan arsitektur *Transformer* dari depan hingga mencapai hasil prediksi. Pada lapisan klasifikasi paling akhir sebelum tebakan kata dihasilkan, *forward pass* pada model *pre-trained* modern termasuk seperti T5 melewati tiga perhitungan utama: Normalisasi, *Linear Layer*, dan *Softmax* [44].

1. Normalisasi Tipe *Root Mean Square*

   Berbeda dengan *Layer Normalization* standar yang menghitung rata-rata, T5 menggunakan *Simplified Layer Normalization* atau RMSNorm untuk mempercepat komputasi. Vektor *Sidden State* masukan $x \in \mathbb{R}^d$ akan diratakan tegangannya tanpa menggeser rata-ratanya menggunakan persamaan:

   $$RMS(x) = \sqrt{\frac{1}{d}\sum_{i=0}^{d} x^2_i + \epsilon} \quad (16)$$

   Setelah nilai scalar RMS didapatkan, vektor akan disesuaikan skalanya dengan parameter bawaan (g) untuk menjaga representasinya:

   $$\bar{x} = \frac{x}{RMS(x)} \odot g \quad (17)$$

2. *Linear Layer*

   Vektor yang telah di stabilkan $\bar{x}$ kemudian diproyeksikan ke ruang dimensi sebesar jumlah kosakata melalui perhitungan *dot product* dengan matriks W:

   $$z = W.\bar{x} + b \quad (18)$$

   Di mana z adalah nilai *logit,* W adalah bobot, dan b adalah parameter bias

3. Fungsi Aktivasi *Softmax*

   Hasil *logit* (z) masih berupa angka desimal acak, model mengunakan fungsi aktivasi *softmax* untuk mengonversinya menjadi distribusi probabilitas dengan rentang 0 hingga 1 yang jumlah totalnya sama dengan 1 [43].

   $$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}} \quad (19)$$

   Di mana $\hat{y}_i$ merupakan nilai probabilitas akhir setiap kandidat kata yang akan dibangkitkan oleh mesin.

2.5.4 *Backpropagation*

Backpropagation adalah algoritma fundamental dalam proses *fine-tuning* yang bertugas memperbarui nilai matriks bobot parameter model agar prediksi menjadi lebih akurat. Proses ini bekerja dengan cara mengirimkan sinyal *error* dari fungsi *loss* berbalik

arah mundur ke arah input jaringan saraf [45]. Prinsip kerja matematis dari *backpropagation* menggunakan Aturan Rantai. Sebagai contoh, untuk lapisan probabilitas *Softmax* dengan fungsi *Cross-Entropy Loss*, gradien turunan kesalahan parsial terhadap setiap nilai *logit* (z_i) dapat disederhanakan menjadi selisih antara nilai prediksi menjadi selisih antara nilai prediksi dengan nilai target aktual.

$$\frac{\partial L_{CE}}{\partial z_i} = \hat{y}_i - y_i \tag{20}$$

Nilai gradien tersebut menunjukkan sejauh mana dan ke arah mana letak kesalahan prediksi mesin. Setelah gradien diketahui, model menggunakan algoritma pengoptimalan (*Optimizer* seperti *Gradient Descent* atau AdamW) untuk merevisi nilai matriks bobot lama (W_{lama}) menjadi bobot yang lebih cerdas (W_{baru}):

$$W_{baru} = W_{lama} - \alpha \left(\frac{\partial L_{CE}}{\partial z_i} \right) \tag{21}$$

Keterangan:

W_{baru} = Nilai bobot baru yang telah diperbaharui

W_{lama} = Nilai matriks bobot sebelumnya

α = *Learning Rate*

$\frac{\partial L_{CE}}{\partial z_i}$ = Gradien fungsi loss terhadap matriks bobot (W)

Melalui siklus berulang *Forward Pass* dan *Backpropagation* ini (iterasi *Epoch*), tingkat *loss* akan terus ditekan turun hingga model bahasa mencapai tingkat kecerdasan yang optimal.

2.5.5 *Fine-tuning*

Fine-tuning merupakan proses penyesuaian model pra-latih terhadap tugas tertentu dengan menggunakan *dataset* yang lebih spesifik. Pada model T5 (*Text-To-Text Transfer Transformer*), proses ini dilakukan dengan tetap menggunakan pendekatan teks-ke-teks, di mana semua tugas dikonversi ke dalam format yang memperlakukan input dan output sebagai urutan teks. Dalam konteks *Automatic Question Generation* (AQG), tugas ini diformulasikan sebagai transformasi teks masukan (seperti paragraf bacaan) menjadi teks keluaran berupa pertanyaan [46]. Melalui proses *fine-tuning* tersebut, model mempelajari pola bahasa, informasi penting dalam teks, serta struktur kalimat tanya yang sesuai dengan konteks. Penyesuaian ini memungkinkan model menghasilkan pertanyaan yang lebih relevan dan selaras dengan isi teks sumber, sehingga pendekatan *fine-tuning* T5 dinilai efektif untuk mendukung sistem pembangkitan pertanyaan otomatis [23].

2.6 Pembelajaran *Adaptif*

Pembelajaran adaptif merupakan metode pembelajaran yang menyesuaikan proses belajar dengan kebutuhan dan kemampuan peserta didik. Pendekatan ini memanfaatkan data untuk mengidentifikasi tingkat pemahaman, kemudian menyediakan materi dan tugas yang disesuaikan agar pembelajaran berlangsung lebih efektif serta mendukung pencapaian tujuan belajar [47]. Selain itu, pembelajaran adaptif juga berupaya menyesuaikan cara penyampaian materi dengan karakteristik dan gaya belajar individu sehingga pengalaman belajar menjadi lebih personal [48]. Penerapan pembelajaran adaptif diharapkan mampu meningkatkan kualitas proses pembelajaran melalui pendekatan yang lebih fleksibel, personal, dan berorientasi pada kebutuhan individu.

2.6.1 Konsep Personalisasi Pembelajaran

Pembelajaran yang dipersonalisasi merupakan suatu pendekatan pendidikan yang berfokus pada penyesuaian proses belajar dengan kebutuhan, minat, serta kemampuan khas setiap peserta didik. Pendekatan ini bertujuan memberikan pengalaman belajar yang disesuaikan bagi setiap siswa dengan memperhatikan kebutuhan individu mereka, sekaligus mengakui adanya perbedaan dalam preferensi, strategi belajar, serta tujuan pembelajaran yang dimiliki masing-masing peserta didik [49]. Penerapan pendekatan ini juga didukung oleh pemanfaatan teknologi pendidikan yang memungkinkan sistem pembelajaran menyesuaikan materi, tingkat kesulitan, metode penyampaian, serta bentuk evaluasi secara adaptif sesuai dengan perkembangan kemampuan siswa. Integrasi pembelajaran yang dipersonalisasi dengan teknologi telah terbukti meningkatkan keterlibatan, motivasi, dan pemahaman materi siswa, sehingga berdampak positif terhadap kinerja dan prestasi akademik mereka [50].

2.6.2 *Initial Ability Test*

Pada penggunaan pertama, sistem belum memiliki informasi mengenai tingkat kemampuan pengguna sehingga belum dapat menentukan soal awal yang sesuai. Oleh karena itu, pengguna terlebih dahulu mengerjakan tes kemampuan awal (*placement test*) untuk memperoleh gambaran mengenai kemampuan yang dimiliki. *Placement test* merupakan bentuk asesmen awal yang digunakan untuk mengidentifikasi tingkat kemampuan peserta sehingga proses pembelajaran maupun penempatan dapat disesuaikan

dengan kemampuan tersebut. Dengan demikian, sistem memperoleh informasi awal yang dapat digunakan sebagai dasar sebelum proses tes adaptif dimulai [51].

Hasil *placement test* digunakan untuk memperoleh estimasi kemampuan awal pengguna sebelum mekanisme adaptif dijalankan. Estimasi kemampuan awal berfungsi sebagai titik awal (*initial ability estimate*) dalam *Computerized Adaptive Testing* sehingga sistem dapat menentukan soal pertama yang sesuai dengan tingkat kemampuan pengguna. Setelah nilai awal diperoleh, estimasi kemampuan selanjutnya diperbarui secara dinamis berdasarkan respons pengguna selama tes adaptif berlangsung [52].

Kemampuan awal ditentukan berdasarkan proporsi jawaban benar yang diperoleh pengguna selama mengerjakan *placement test*. Jika pengguna menjawab benar sebanyak r dari total n soal, maka proporsi jawaban benar dapat dihitung dengan Persamaan (22).

$$p = \frac{r}{n} \tag{22}$$

Keterangan:

p = proporsi jawaban benar,

r = jumlah jawaban benar,

n = jumlah soal pada tes kemampuan awal

Nilai proporsi jawaban benar berada pada rentang 0 hingga 1 sehingga kurang sesuai digunakan sebagai nilai awal dalam proses adaptasi. Oleh karena itu, nilai tersebut ditransformasikan menggunakan fungsi logit (*log-odds*) agar diperoleh skor dengan rentang nilai yang lebih luas, yaitu dari $-\infty$ hingga $+\infty$. Transformasi ini menghasilkan nilai awal yang merepresentasikan kecenderungan kemampuan pengguna berdasarkan hasil *placement test* [53]. Persamaan transformasi logit ditunjukkan pada Persamaan (23).

$$\theta_{awal} = \ln\left(\frac{p}{1-p}\right) \tag{23}$$

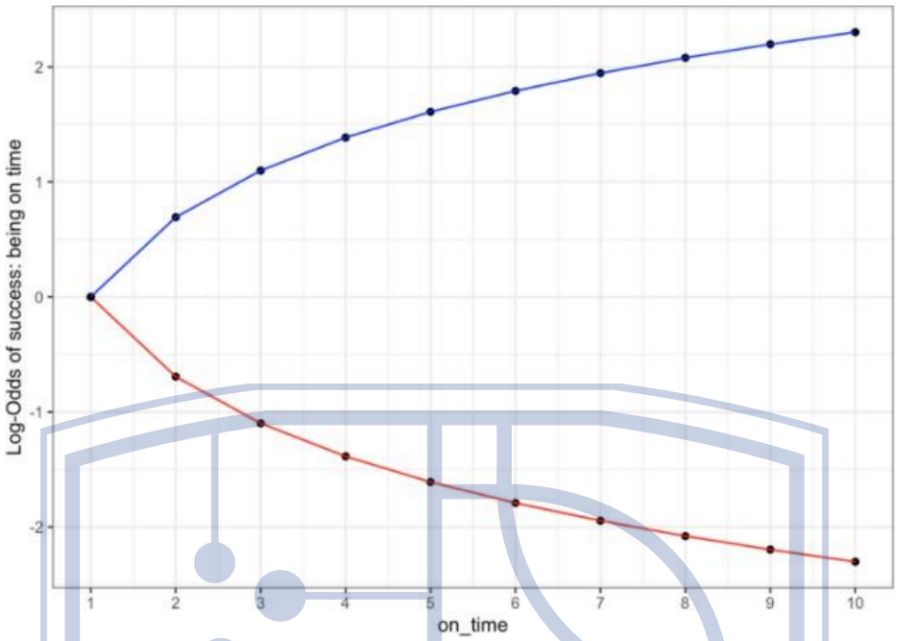
Keterangan:

θ_{awal} = Estimasi kemampuan laten awal

p = proporsi jawaban benar

ln = *logaritma natural*

Hasil transformasi logit memiliki penjelasan yang sederhana. Nilai positif menunjukkan bahwa proporsi jawaban benar lebih besar dari 50%, sedangkan nilai negatif menunjukkan bahwa proporsi jawaban benar kurang dari 50%. Apabila proporsi jawaban benar sama dengan 50%, maka nilai logit bernilai nol. Ilustrasi perubahan proporsi jawaban benar menjadi nilai *log-odds* ditunjukkan pada Gambar 2.5.



Gambar 2.5 Transformasi Logit dari Proporsi Jawaban Benar [54].

Nilai kemampuan awal yang diperoleh pada tahap ini hanya digunakan sebagai inisialisasi untuk menentukan soal pertama yang diberikan kepada pengguna. Setelah tes adaptif dimulai, estimasi kemampuan pengguna tidak lagi menggunakan transformasi *logit*. Proses estimasi kemampuan selanjutnya dilakukan secara dinamis menggunakan *Item Response Theory* (IRT) yang dikombinasikan dengan *Elo Rating* berdasarkan respons pengguna pada setiap butir soal. Dengan demikian, transformasi *logit* hanya berfungsi sebagai langkah awal untuk mengatasi *cold-start problem*, sedangkan pembaruan kemampuan selama tes sepenuhnya ditangani oleh mekanisme IRT dan Elo Rating.

2.6.3 *Item Response Theory* (IRT)

Item Response Theory (IRT) adalah kerangka kerja evaluasi matematis yang digunakan untuk menganalisis hubungan antara kemampuan laten (tersembunyi) seseorang dengan probabilitasnya dalam menjawab suatu butir soal secara benar [55]. Berbeda dengan Teori Tes Klasik, IRT memiliki keunggulan berupa sifat invarian. Artinya, parameter tingkat kesulitan butir soal tidak bergantung pada distribusi kemampuan kelompok peserta tes, dan sebaliknya, estimasi kemampuan peserta tes tidak bergantung pada paket soal spesifik yang dikerjakan. Sifat ini menjadikan pengukuran dengan IRT jauh lebih objektif, konsisten, dan tidak menimbulkan ketidakadilan dalam proses pengujian pada berbagai kelompok pengguna [56].

Salah satu model fundamental dalam IRT adalah Model 1-Parameter Logistic (1PL) atau yang sering disebut sebagai Model *Rasch*. Model ini mengasumsikan bahwa probabilitas peserta menjawab benar hanya dipengaruhi oleh satu karakteristik butir soal, yaitu tingkat kesulitan (b). Secara matematis, peluang peserta dengan tingkat kemampuan (θ) untuk menjawab benar pada butir soal dirumuskan sebagai berikut [12]:

$$P(\theta) = \frac{1}{1+e^{-(\theta-b)}} \tag{24}$$

Keterangaan:

$P(\theta)$ = probabilitas peserta menjawab benar

θ = kemampuan peserta

b = tingkat kesulitan soal

e = bilangan eksponensial ($\approx$ 2,718)

Pada implementasi *Item Response Theory*, parameter tingkat kesulitan butir soal umumnya diperoleh melalui proses kalibrasi berdasarkan respons peserta tes. Namun, penelitian ini tidak melakukan proses kalibrasi karena berfokus pada pengembangan mekanisme *adaptive testing* serta memanfaatkan bank soal yang telah memiliki kategori tingkat kesulitan. Oleh karena itu, kategori kesulitan yang tersedia pada dataset digunakan sebagai dasar untuk merepresentasikan parameter kesulitan butir soal dalam model Rasch. Pendekatan ini memungkinkan model IRT digunakan sebagai mekanisme perhitungan probabilitas jawaban benar tanpa memerlukan proses estimasi parameter butir secara empiris [57].

2.6.4 *Elo Rating System (ERS)*

Meskipun perhitungan IRT tradisional memberikan probabilitas yang objektif, penggunaannya secara *real-time* menuntut evaluasi ulang terhadap seluruh riwayat jawaban pengguna, yang berpotensi memberikan beban tinggi pada komputasi memori server. Oleh karena itu, probabilitas ekspektasi dari model Rasch 1PL dikombinasikan dengan metode *Elo Rating System* untuk menciptakan pendekatan iteratif yang efisien [13]. Pendekatan ini memungkinkan estimasi kemampuan diperbarui lebih cepat yang berdasarkan respons pada butir soal terakhir. Perhitungan pembaruan kemampuan dirumuskan sebagai berikut:

$$\theta_{baru} = \theta_{lama} + K(S - P(\theta)) \tag{25}$$

Keterangan:

θ_{baru} = Kemampuan baru

θ_{lama} = Kemampuan lama

K = Parameter *learning rate*

S = Skor respons aktual (1 jika benar; 0 jika salah)

$P(\theta)$ = Probabilitas ekspektasi sistem

2.7 Evaluasi ROUGE

Evaluasi merupakan proses yang dilakukan secara sistematis untuk mengetahui apakah tujuan yang telah ditetapkan berhasil dicapai. Proses ini mencakup serangkaian kegiatan penilaian yang dilakukan secara terstruktur dan menyeluruh dengan berpedoman pada tujuan yang jelas [58]. Dalam konteks penelitian ini, evaluasi dilakukan dengan menilai kinerja model melalui berbagai indikator yang mengukur tingkat ketepatan, konsistensi, dan kesesuaiannya terhadap tugas yang dijalankan [59].

Evaluasi kualitas teks pada sistem *Natural Language Processing* (NLP) dapat dilakukan menggunakan metrik otomatis *Recall-Oriented Understudy for Gisting Evaluation* (ROUGE). Metrik tersebut digunakan untuk menilai sejauh mana teks yang dihasilkan sistem memiliki kemiripan dengan teks acuan yang dianggap benar atau sesuai [60]. ROUGE menilai kesamaan dengan mengukur tumpang tindih kata n-gram antara ringkasan otomatis dan ringkasan referensi [61].

Dalam penelitian ini, evaluasi dilakukan menggunakan ROUGE-1, ROUGE-2, dan ROUGE-L. ROUGE-1 mengukur kesamaan berdasarkan unigram, ROUGE-2 berdasarkan bigram, sedangkan ROUGE-L menggunakan pendekatan *Longest Common Subsequence* (LCS) untuk membandingkan urutan kata antara teks hasil sistem dan teks referensi. Nilai LCS menunjukkan panjang urutan kata terpanjang yang sama, sedangkan *M* merupakan jumlah kata dalam teks referensi.

$$ROUGE - L = \frac{LCS}{M} \tag{26}$$

Setiap nilai ROUGE dievaluasi menggunakan tiga metrik utama, yaitu *recall*, *precision*, dan *F1-score*. *Recall* mengukur jumlah kata yang berhasil ditangkap dari teks referensi, *precision* mengukur ketepatan kata yang dihasilkan oleh sistem, dan *F1-score* merupakan kombinasi keduanya.

$$Recall = \frac{W_{overlapped}}{W_{ref}} \tag{27}$$

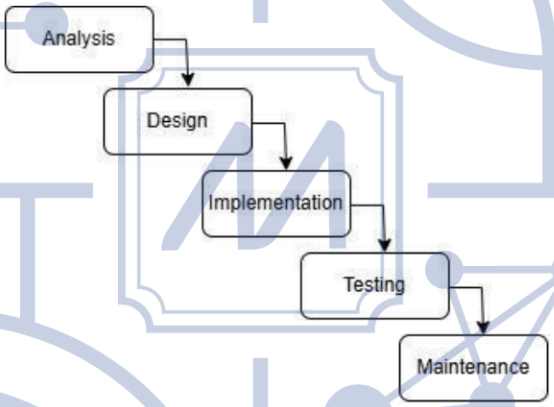
$$Precision = \frac{W_{overlap}}{W_{ref}} \tag{28}$$

$$F1 - Score = \frac{2\,x\,Recall\,x\,Precision}{(Recall + Precision)} \tag{29}$$

Dengan demikian, ROUGE digunakan untuk mengukur tingkat kemiripan antara teks yang dihasilkan oleh sistem dengan teks referensi. Semakin tinggi nilai ROUGE yang diperoleh, maka semakin baik kualitas teks yang dihasilkan oleh sistem [62]. Nilai ROUGE yang diperoleh sistem kemudian dapat dikategorikan terhadap empat tingkatan berdasarkan penelitian terdahulu yaitu ≥ 0,75 (Sangat Baik), 0,50 s/d 0,74 (Baik), 0,25 s/d 0,49 (Cukup) dan < 0,25 (Kurang) [63][64].

2.8 Metodologi *Waterfall*

Metodologi *Waterfall* atau model air terjun adalah model pengembangan perangkat lunak yang dilakukan secara sistematis dan berurutan. Tahapan dalam metode ini meliputi [65]:



Gambar 2.6 Metodologi *Waterfall* [65].

1. *Analysis:* Pada tahap ini dilakukan analisis terhadap kebutuhan sistem, yang mencakup kebutuhan fungsional maupun kebutuhan non fungsional.
2. *Design:* Tahap perancangan merupakan tahap lanjutan yang berfokus pada penyusunan desain sistem, meliputi perancangan antarmuka pengguna serta struktur basis data yang akan digunakan dalam aplikasi.
3. *Implementation:* Tahap ini merupakan fase implementasi, di mana proses pengkodean mulai dilakukan. Pada tahap ini, desain basis data dan desain antarmuka yang telah dirancang sebelumnya direalisasikan menggunakan DBMS serta bahasa pemrograman yang sesuai.
4. *Testing:* Tahap pengujian merupakan fase di mana sistem yang telah dikembangkan mulai diuji coba untuk memastikan kesesuaiannya dengan kebutuhan yang telah

ditetapkan. Proses ini bertujuan untuk mengidentifikasi dan meminimalkan kesalahan agar sistem dapat berfungsi dengan baik dan layak digunakan.

5. *Maintenance:* Tahap ini merupakan fase pemeliharaan, yaitu tahap setelah sistem selesai dikembangkan dan digunakan. Pemeliharaan bertujuan untuk memperbaiki kesalahan yang ditemukan setelah implementasi, menyempurnakan fungsi sistem, serta meningkatkan kinerja agar sistem tetap optimal dan sesuai dengan kebutuhan pengguna.

2.9 Pengujian *Black-Box*

Proses pengujian merupakan tahapan krusial dalam siklus pengembangan perangkat lunak demi menjamin kualitas produk dan memastikan seluruh fitur berjalan sesuai ekspektasi pengguna. Melalui pengujian ini, pengembang dapat melacak *bug* atau celah kesalahan, memvalidasi kinerja setiap fungsi, serta mengukur performa aplikasi sebelum akhirnya benar-benar dirilis ke public. Agar hasilnya optimal, perancangan strategi uji harus mempertimbangkan banyak aspek secara matang, mulai dari target pencapaian, manajemen waktu, ketersediaan sumber daya, hingga kesiapan lingkungan uji itu sendiri. Walaupun agenda pengujian ini belum tentu bisa menjamin sebuah aplikasi bebas dari cacat secara mutlak, langkah ini sangat efektif untuk membangun rasa aman dan meningkatkan kepercayaan pengguna saat mengoperasikan sistem.

Dalam praktiknya, terdapat beberapa skema pengujian yang biasa diterapkan, yakni *black-box*, *white-box*, dan *grey-box testing*. Meski begitu, fokus pengujian biasanya bertumpu pada dua pendekatan utama, yaitu *black-box* dan *white-box testing* [66]. *Black-box* testing merupakan metode pengujian perangkat lunak yang berfokus sepenuhnya pada spesifikasi fungsional sistem tanpa melibatkan analisis terhadap struktur internal atau kode program. Hal ini berbeda dengan *white-box testing* yang justru mewajibkan pemeriksa untuk membedah logika pemrograman serta melacak setiap jalur kode di dalam aplikasi. Kedua metode ini menyimpan karakteristik, kelebihan, serta keterbatasannya masing-masing. Metode *black-box* umumnya terasa lebih praktis dan efisien ketika skenario uji (*test cases*) yang ditangani berjumlah terbatas. Namun, apabila sistem memerlukan intensitas dan frekuensi pengujian yang tinggi secara berulang, maka pendekatan *white-box* menjadi pilihan yang jauh lebih tepat [66].