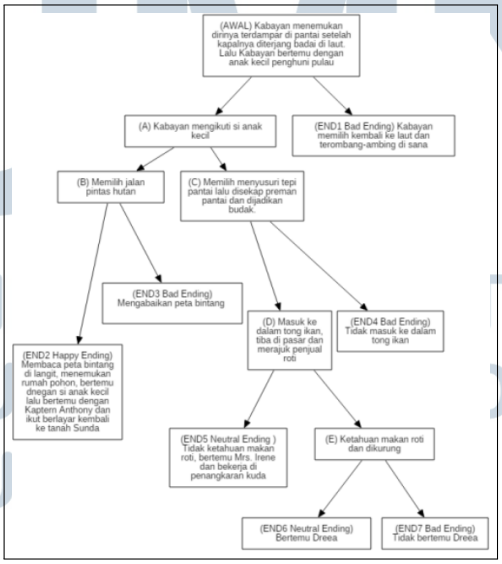
BAB 2
LANDASAN TEORI

2.1 Visual Novel

Visual novel merupakan media penyajian cerita digital yang dituliskan dalam teks. *Visual novel* memiliki latar belakang gambar serta ilustrasi karakter yang berubah sesuai situasi dan kondisi cerita. *Visual novel* juga dapat didukung oleh elemen suara seperti latar belakang musik, efek suara, hingga *voice acting* dialog karakter [10].

Visual novel dapat menyediakan cerita interaktif dengan memberikan pilihan pada pemain. Alur cerita dapat berubah antar pemain berdasarkan pilihan-pilihan yang diberikan masing-masing pemain. Sehingga *visual novel* dapat memberikan cerita bervariasi hingga akhir cerita [10].

Contoh percabangan alur cerita dapat terlihat pada jurnal ”PERANCANGAN GAME VISUAL NOVEL ”THE ADVENTURE OF KABAYAN” SEBAGAI MEDIA BELAJAR BAHASA INGGRIS UNTUK TOEFL”. Gambar 2.1 mengilustrasikan pohon cerita yang diambil dari jurnal tersebut, di mana alur cerita berkembang dari atas ke bawah. Percabangan pada pohon cerita tersebut menunjukkan skenario cerita yang ditentukan oleh pilihan pemain [10].



Gambar 2.1. Contoh percabangan alur cerita visual novel

2.2 Generative AI

Generative AI merupakan cabang dari kecerdasan buatan (AI) yang fokus dalam pembuatan konten yang tampak baru dan bermakna, seperti dalam bentuk teks, gambar, atau suara [7].

Generative AI mampu membuat konten baru berdasarkan pola-pola yang didapatkan dalam data *training*. Contoh *Generative AI* adalah LLM seperti GPT, LLaMA, Gemma dan model *text-to-image* seperti *Stable Diffusion*. LLM menciptakan konten baru dengan melanjutkan sebuah karya teks [11]. Sementara *Stable Diffusion* menciptakan konten gambar dengan proses *diffusion* yang dipandu oleh beberapa kata kunci [12].

2.2.1 Transformer

Transformer adalah salah satu jenis arsitektur *neural network*. *Transformer* umum dipakai dalam LLM karena kemampuannya dalam memproses data secara paralel. Dalam konteks teks, *Recurrent Neural Networks* (RNN) menganalisis elemen-elemen teks satu-per-satu, sementara *Transformer* dapat memecah elemen-elemen teks menjadi berbagai *token* yang dapat diproses sekaligus oleh mekanisme *Attention* untuk mendapatkan prediksi *token* berikutnya. Arsitektur *Transformer* memecahkan data menjadi *token* untuk diproses secara paralel, parralelisme ini membuat LLM menjadi *scalable* [13].

Implementasi *Transformer* dalam berbagai model dapat bervariasi. Variasi-variasi ini dinamakan berdasarkan bagian *Transformer* yang terpakai, misalnya *encoder-decoder model* atau *decoder-only model*. Proses *encoding* adalah proses perubahan data menjadi *token*, sementara proses *decoding* merupakan proses perubahan *token* menjadi data [13].

A Token

Sebuah *token*, dalam konteks *Transformer*, merupakan sebuah vektor angka yang berdimensi besar. Setiap *token* dalam *Transformer* memiliki makna abstrak yang terdiri dari masing-masing angka. Sebuah pasang *token* dapat memiliki korelasi positif atau negatif berdasarkan hasil *dot product* dari kedua vektor. Sekumpulan *token* dapat direpresentasikan oleh matriks, sementara operasi yang dilakukan oleh proses *Encoding*, *Decoding*, dan mekanisme *Attention* dapat

direpresentasikan oleh perkalian matriks yang menggunakan matriks sekumpulan *token* tersebut [13].

B Encoding dan Decoding

Proses *Encoding* merupakan proses perubahan data menjadi *Token*, dan proses *Decoding* merupakan sebaliknya. Data yang dimaksud dapat berupa teks, gambar, suara, dan lain-lain. Data yang di*token*asikan menghasilkan *token-token* yang berhubungan dengan jenis data yang dipakai, misalnya untuk data teks maka *token* berupa potongan teks, untuk data gambar maka *token* berupa potongan gambar, dan seterusnya [13].

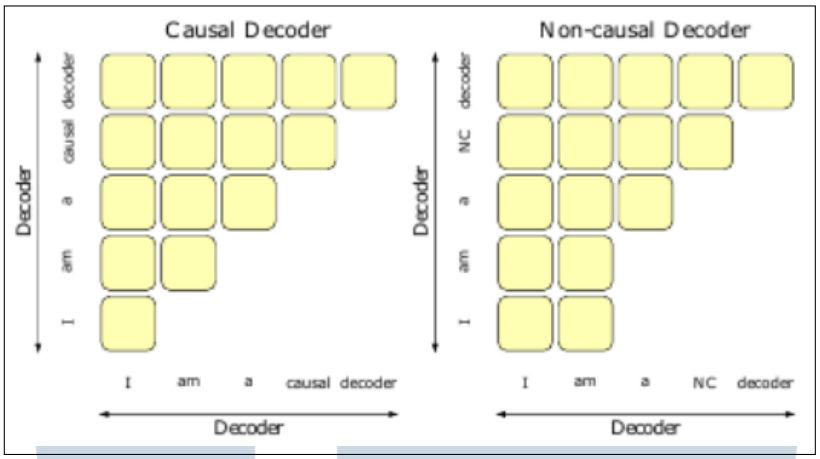
Semua *token* yang mungkin dipakai oleh model telah ditetapkan pada awal pembuatan model, sebuah ”kamus” *token* berbentuk matriks yang dipakai oleh *encoder* dan *decoder* sebagai dasar *tokenasi*. Misalnya dalam konteks teks, sebuah *token* dapat mewakili sebuah huruf, potongan kata, sebuah kata utuh, angka, atau simbol. Angka-angka dalam Matriks *encoder* serta Matriks *decoder* ditentukan setelah melalui proses pelatihan [13].

2.2.2 Attention

Token-token yang telah melalui proses *token*asi masuk dalam proses mekanisme *Attention*. Mekanisme *Attention* adalah proses yang diulang-ulang pada sekumpulan *token*, untuk mendistribusi informasi konteks *token* pada setiap *token* lainnya. Proses ini berupa *perkalian* matriks antara matriks *token* dengan sebuah matriks *weight* yang dibuat dari proses *training*. Melalui proses *Attention*, sebuah *token* yang hanya merepresentasikan sebuah potongan data dapat menyerap konteks dari potongan-potongan data lainnya [13].

Terdapat dua jenis proses penyerapan konteks yang dapat dilatih dalam mekanisme *Attention*, yaitu *Causal Decoder* dan *Prefix Decoder*. Dalam *Causal Decoder*, setiap *token* memiliki urutan, dan penyerapan konteks masing-masing *token* hanya terjadi dari *token* dengan urutan berdahulu. *Causal Decoder* berguna untuk membuat model yang dapat menciptakan teks dengan berurut. Dalam *Prefix Decoder* atau juga disebut sebagai *Non-causal Decoder*, konteks diserap secara dua arah. *Prefix Decoder* berguna untuk membuat model yang dapat melengkapi sebuah gambar atau dokumen menggunakan konteks data sebelum dan sesudah bagian yang terpotong [14]. Perbedaan penyerapan konteks antara *Causal Decoder dan*

Prefix Decoder diilustrasikan dalam Gambar 2.2.



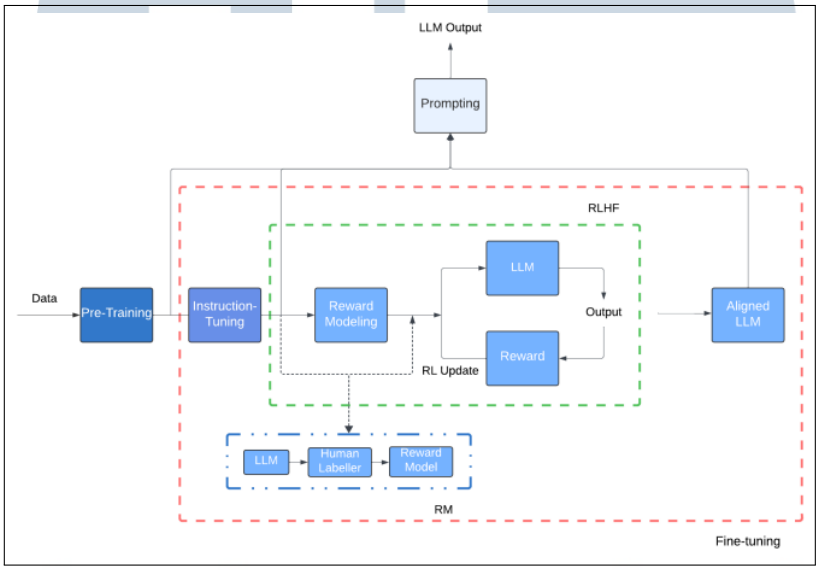
Gambar 2.2. Perbedaan penyerapan konteks dalam *Causal Decoder* dan *Prefix Decoder*

2.2.3 Large Language Model

Language Model (LM) adalah sebuah model yang dilatih untuk menyelesaikan sebuah *Language Task* seperti terjemahan, rangkuman, pengambilan informasi, atau interaksi percakapan. LM yang mampu melakukan pelatihan secara *self-supervised* disebut sebagai *Pre-trained Language Model* (PLM). PLM mampu dilatih dalam kumpulan data teks yang lebih besar dari LM yang dilatih dalam lingkungan *supervised*. Performa dalam PLM mudah bertambah berdasarkan jumlah parameter dalam model serta jumlah data pelatihan. *Large Language Model* (LLM) adalah sebuah PLM dengan jumlah parameter sebesar puluhan hingga ratusan miliar parameter, dan dilatih menggunakan data sebesar berbagai *Gigabyte* hingga *Terabyte*. Transisi dari PLM menjadi LLM terjadi dengan dukungan penerbitan arsitektur *Transformer* dan ketersediaan data pelatihan berskala besar [14].

Terdapat berbagai jenis model LLM yang dibedakan berdasarkan cara penggunaan mekanisme *Attention* dalam *Transformer* serta berdasarkan jenis pelatihan yang diterapkan. Proses awal pelatihan model disebut sebagai *pre-training*, sementara pelatihan selanjutnya disebut sebagai *fine-tuning*. Kemampuan utama dalam model LLM ditentukan oleh *pre-training*, misalnya kemampuan menciptakan teks berdasarkan sebuah *prefix* merupakan jenis *pre-training Prefix Language Modeling*, sementara kemampuan melengkapi teks berdasarkan data sebelum dan setelah letak target pengisian teks merupakan jenis *pre-training Masked Language Modeling*. Setelah *pre-training*, sebuah model LLM dapat

membuat teks, namun untuk penggunaan yang lebih spesifik model LLM dapat melalui proses *fine-tuning*. Contoh dari *fine-tuning* adalah *Instruction-tuning* serta *Alignment-tuning*. Proses *Instruction-tuning* dapat membantu LLM untuk mengikuti sebuah instruksi, sementara Proses *Alignment-tuning* dilakukan untuk mengurangi hasil teks yang berbahaya [14]. Proses pelatihan model LLM diilustrasikan dalam Gambar 2.3 RL merupakan singkatan dari *Reinforcement Learning*, RM merupakan singkatan dari *Reward Modeling*, RLHF merupakan singkatan dari *Reinforcement Learning with Human Feedback*.



Gambar 2.3. Diagram Proses Pelatihan LLM

2.3 Prompt Engineering

Prompt engineering adalah teknik penggunaan *Language Model* menggunakan pola-pola *prompt* yang dapat menambahkan kemampuan *Language Model* dalam menyelesaikan tugas tertentu. *Prompt engineering* dapat dipakai untuk memitigasi limitasi LLM, seperti: tidak terdapat ingatan yang kuat, *output* yang tidak konsisten dari *input* yang sama, serta fabrikasi konten yang tidak benar seperti halusinasi [15].

Salah satu contoh *prompt engineering* adalah *prompt chaining*, artinya membagi suatu tugas menjadi berbagai sub-tugas, lalu hasil *output* dari suatu sub-tugas dipakai menjadi *input* untuk sub-tugas lain. *Prompt chaining* dapat membuat LLM lebih transparan dan *reliable*, serta memudahkan analisis jika LLM tidak berhasil menjalankan tugasnya [16].