NO: 01022127/INF/2022

RINGKASAN EKSTRAKTIF OTOMATIS PADA BERITA BERBAHASA INDONESIA MENGGUNAKAN METODE BERT

SKRIPSI

Diajukan untuk memenuhi persyaratan penyelesaian program S-1
Program Studi Informatika Fakultas Teknologi Industri
Universitas Kristen Petra

Oleh :
Franky Halim
NRP: C14170033

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS KRISTEN PETRA
SURABAYA
2022

LEMBAR PENGESAHAN

SKRIPSI

RINGKASAN EKSTRAKTIF OTOMATIS PADA BERITA BERBAHASA INDONESIA MENGGUNAKAN METODE BERT

Oleh :

Franky Halim NRP : C14170033

Diterima Oleh :
Program Studi Informatika
Fakultas Teknologi Industri
Universitas Kristen Petra

Surabaya, 10 Januari 2022

Pembimbing I	Pembimbing II
LILIANA, Ph.D.	Ir. KARTIKA GUNADI, M.T.
NIP: 03-024	NIP: 88-004

Ketua Tim Penguji

LEO WILLYANTO SANTOSO, M.IT.
NIP: 03-023

Kepala Program Studi

HENRY NOVIANUS PALIT, Ph.D.
NIP: 14-001

LEMBAR PERNYATAAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Sebagai mahasiswa Universitas Kristen Petra, yang bertanda tangan dibawah ini, saya:

Nama : Franky Halim

NRP : C14170033

Demi mengembangkan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Kristen Petra Hak Bebas Royalti Non-Eksklusif (*Non- Exclusive Royalti-Free Rights*) atas karya ilmiah saya yang berjudul: Ringkasan Ekstraktif Otomatis pada Berita Berbahasa Indonesia Menggunakan Metode BERT. Dengan Hak Bebas Royalti Non-Eksklusif ini Universitas Kristen Petra berhak menyimpan, mengalih-media format-kan, mengelolanya dalam bentuk pangkalan data (*database*), mendistribusikannya dan menampilkan/ mempublikasikannya di internet atau media lain untuk kepentingan akademis tanpa perlu meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis/pencipta.

Saya bersedia untuk menanggung secara pribadi, tanpa melibatkan pihak Universitas Kristen Petra, segala bentuk tuntutan hukum yang timbul atas pelanggaran Hak Cipta dalam karya ilmiah saya ini.

Demikian surat pernyataan ini saya buat dengan sebenarnya.

Surabaya, 10 Januari 2022

Yang menyatakan,

Franky Halim

KATA PENGANTAR

Segala puji dan syukur kepada Tuhan Yesus Kristus yang selalu dengan setia dan penuh kasih menyertai, membimbing, memberkati, dan memberikan kekuatan bagi peneliti hingga dapat menyelesaikan tugas akhir dengan baik. Ketika peneliti merasa tidak sanggup, tangan Tuhan yang selalu menopang dan memberikan hikmat yang luar biasa. Peneliti sangat bersyukur dan berterima kasih pada Tuhan atas setiap proses pengerjaan tugas akhir ini.

Peneliti juga berterima kasih pada setiap orang yang selalu menemani, mendengarkan, memberikan semangat, dan membimbing peneliti khususnya selama mengenyam pendidikan di Universitas Kristen Petra. Terima kasih karena sudah hadir dan memberikan warna yang beragam di dalam kehidupan perkuliahan peneliti. Orang-orang yang berperan besar sehingga dapat terselesaikannya tugas akhir ini, antara lain:

1. Ibu Liliana, Ph.D., selaku dosen pembimbing I, yang telah memberikan arahan, motivasi serta meluangkan waktu selama proses pembuatan skripsi berlangsung.
2. Bapak Ir. Kartika Gunadi, M.T., selaku dosen pembimbing II, yang telah memberikan arahan, motivasi serta meluangkan waktu selama proses pembuatan skripsi berlangsung.
3. Ibu Silvia Rostianingsih, M.MT., selaku Koordinator Skripsi Program Teknik Informatika dan Sistem Informasi Bisnis Universitas Kristen Petra.
4. Segenap dosen dan staff pengajar di Program Studi Teknik Informatika dan Sistem Informasi Bisnis Universitas Kristen Petra.
5. Keluarga yang telah banyak memberikan dukungan doa dan motivasi hingga penulis mampu menyelesaikan tugas akhir guna meraih gelar kesarjanaan ini.
6. Antonius Tanuwijaya, Edward Hosea, Jehezkiel Hardwin Tandijaya, Reyner, dan teman-teman lain yang tidak dapat disebutkan satu per satu yang telah memberikan banyak dukungan, motivasi, dan semangat untuk peneliti.
7. Pihak-pihak lain yang telah memberikan bantuan secara langsung maupun tidak langsung dalam pembuatan tugas akhir ini yang tidak dapat disebutkan satu per satu.

Peneliti menyadari bahwa penulisan skripsi ini masih jauh dari sempurna. Oleh karena itu, peneliti mengharapkan segala petunjuk, kritik, dan saran yang membangun dari pembaca agar dapat menunjang pengembangan dan perbaikan penulisan.

Akhir kata, peneliti mohon maaf apabila ada kekurangan dalam penulisan tugas akhir ini dan peneliti dengan senang hati menerima saran dan kritik yang membangun dari pembaca.

Surabaya, 10 Januari 2022

Peneliti

ABSTRAK

Franky Halim:

Skripsi

Ringkasan Ekstraktif Otomatis pada Berita Berbahasa Indonesia Menggunakan Metode BERT

Pada era yang semakin modern, informasi menjadi bagian yang penting dalam kehidupan sehari-hari. Dalam mendapatkan informasi terdapat beberapa hal yang dapat dilakukan dimana salah satunya adalah dengan membaca. Dengan semakin banyaknya informasi yang tersedia di internet dapat membuat manusia kerepotan untuk terus mengikuti perkembangannya. Berita *online* juga merupakan salah satu sumber informasi yang ada di internet dengan jumlah yang sangat banyak serta topik yang beragam. Membaca kesuluruhan informasi tersebut terkadang juga memerlukan waktu yang lama. Oleh karena itulah, diperlukan suatu pembuatan ringkasan dari informasi berita yang tersedia secara *online* untuk mengurangi waktu baca dan mendapatkan informasi yang relevan.

Pada penelitian ini akan dilakukan pembuatan ringkasan berita dengan memilih kalimat penting dari teks berita. Metode yang digunakan adalah *Bidirectional Encoder Representations from Transformers* dengan tambahan *transformer encoder layer.*

Berdasarkan hasil pengujian yang telah dilakukan, *pre-trained* model indolem/indobert-base-uncased dapat menghasilkan nilai *F1-Score* terbaik untuk ROUGE-1 sebesar 57.17, ROUGE-2 sebesar 51.27, dan ROUGE-L sebesar 55.20 pada referensi abstraktif serta ROUGE-1 sebesar 84.46, ROUGE-2 sebesar 83.21, dan ROUGE-L sebesar 83.40 pada referensi ekstraktif.

Kata kunci:

Text Summarization, *Online News, Bidirectional Encoder Representations from Transformers*.

ABSTRACT

Franky Halim:

Undergraduate Thesis

Automatic Extractive Summarization on Indonesian News using BERT Method

In this modern era, information has become an important part of everyday life. In getting information several things can be done where one of them is by reading. With the increasing amount of information available on the internet, it is difficult for humans to keep abreast of developments. Online news is also one of the sources of information on the internet with a very large number and various topics. Reading the whole information sometimes also takes a long time. Therefore, it is necessary to make a summary of the available online news to reduce reading time and obtain relevant information.

In this research, a summary of the news will be made by selecting important sentences from the news text. The method used in this research is Bidirectional Encoder Representations from Transformers with the addition of transformer encoder layer.

Based on the results of the tests that have been carried out, the pre-trained indolem/indobert-base-uncased model can produce the best *F1-Score* 57.17 for ROUGE-1, 51.27 for ROUGE-2, and 55.20 for ROUGE-L using abstractive reference and 84.46 for ROUGE-1, 83.21 for ROUGE-2, and 83.40 for ROUGE-L using extractive reference.

Keyword:

Text Summarization, *Online News, Bidirectional Encoder Representations from Transformers*.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR PERSAMAAN

DAFTAR TABEL

DAFTAR SEGMEN DATA

1. PENDAHULUAN

1.1 Latar Belakang Permasalahan

Di era modern ini, informasi merupakan bagian yang penting dalam kehidupan sehari-hari. Dalam mendapatkan informasi terdapat beberapa hal yang dapat dilakukan dimana salah satunya adalah dengan membaca. Suatu informasi dapat diperoleh dengan berbagai cara seperti mengakses *web*. Akan tetapi, semakin banyaknya informasi yang ada di internet membuat manusia kerepotan untuk terus mengikuti perkembangannya (El-Kassas et al., 2020). Maka dari itu, diperlukan suatu pembuatan ringkasan terhadap informasi-informasi yang tersedia secara *online*.

Ringkasan merupakan suatu teks singkat yang dihasilkan dari kumpulan teks panjang namun tetap menyimpan informasi penting dari teks asalnya (Joshi et al., 2019). Dengan bantuan ringkasan akan membantu dalam membaca dengan waktu yang lebih cepat dibandingkan dengan harus membaca informasi secara utuh. Selain itu, ringkasan juga membantu dalam mengabaikan informasi yang tidak relevan tanpa harus kehilangan makna dari informasi yang dibaca. Dalam melakukan pembuatan ringkasan juga terkadang masih dilakukan secara manual oleh manusia yang memakan waktu lama karena jumlah informasi yang ada sangat banyak. Sehingga pembuatan ringkasan secara otomatis diperlukan untuk mengatasi hal ini.

Berita *online* merupakan salah satu sumber informasi yang ada di internet dengan jumlah yang sangat banyak serta topik yang beragam (Schmitt et al., 2017). Topik berita dapat meliputi mengenai politik, ekonomi, olahraga, teknologi, dan masih banyak lagi. Adanya variasi informasi yang luas dan sering digunakan sehingga membuat berita menjadi objek pada penelitian ini. Kemudian, pada penelitian ini dilakukan pada berita berbahasa Indonesia karena untuk pembuatan ringkasan secara otomatis pada berita berbahasa Indonesia belum terlalu berkembang seperti berita berbahasa Inggris. Selain itu, bahasa Indonesia juga masih tergolong ke dalam *low-resource language* bila dibandingkan dengan *high-resource language* seperti bahasa Inggris sehingga perlu dilakukan penelitian untuk pengembangan *natural language processing* berbahasa Indonesia (Hirschberg & Manning, 2015).

Pada model *Bidirectional Encoder Representations from Transformers* (BERT), dilakukan penerapan model secara dua arah dengan menggunakan *Masked LM* (MLM) dengan melihat konteks dari kalimat untuk prediksi terhadap kalimat yang sudah di-*mask*. Lalu, BERT bergantung

juga pada *transformer* yang merupakan mekanisme *attention* yang mempelajari relasi kontekstual antar kata dalam teks. Pada BERT menerima input berupa kalimat yang diubah terlebih dahulu menjadi *sequence tokens* dan diproses dalam *transformer* (Devlin et al., 2019).

Terdapat beberapa penelitian yang telah melakukan peringkasan secara otomatis pada berita *online* berbahasa Inggris dengan menggunakan *pre-trained encoder* BERT. Pada penelitian yang dilakukan dengan BERT digunakan tambahan susunan *transformer encoder* setelah dihasilkan *embedding* dari BERT. Evaluasi terhadap model yang diusulkan mampu menghasilkan ringkasan ekstraktif berita dengan skor yang paling tinggi bila dibandingkan dengan metode lainnya (Liu & Lapata, 2019).

Pada penelitian sebelumnya yang dilakukan oleh Kristian Halim (Halim et al., 2020), ringkasan ekstraktif yang dihasilkan dengan menggunakan *recurrent neural network* mampu mencapai skor ROUGE terbaik sekitar 80% dengan referensi ekstraktif dan 50% dengan referensi abstraktif. Berdasarkan pada penelitian yang telah dilakukan oleh Liu dan Lapata, untuk mengetahui penerapan dan perbandingan metode BERT dengan metode lain seperti *recurrent neural network* diperoleh bahwa metode BERT menghasilkan skor yang paling tinggi (Liu & Lapata, 2019). Selain itu, BERT juga dapat mempelajari kata yang tidak dikenali dengan mengubahnya menjadi sub-kata. Maka dari itu, pada penelitian yang akan dilakukan ini akan menggunakan BERT dalam menghasilkan ringkasan ekstraktif pada berita berbahasa Indonesia.

Dalam pembuatan ringkasan otomatis berdasarkan pada hasilnya dapat dibedakan menjadi dua yaitu abstraktif dan ekstraktif (El-Kassas et al., 2020). Pada penelitian ini akan digunakan metode ekstraktif dalam pembuatan ringkasannya karena pengguna internet lebih memilih teks ringkasan yang mirip dengan teks asli yang dibuat oleh penulis (Schmitt et al., 2017). Metode yang akan digunakan dalam penelitian ini untuk menghasilkan ringkasan ekstraktif adalah *Bidirectional Encoder Representations from Transformers* (BERT).

1.2 Perumusan Masalah

Berdasarkan dari latar belakang yang ada, dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana performa dari *Bidirectional Encoder Representations from Transformers* dalam melakukan peringkasan ekstraktif otomatis pada berita Berbahasa Indonesia ?
2. Seberapa besar perbedaan skor hasil ringkasan ekstraktif bila dibandingkan dengan referensi ekstraktif dan abstraktif dengan menggunakan *Bidirectional Encoder Representations from Transformers* ?

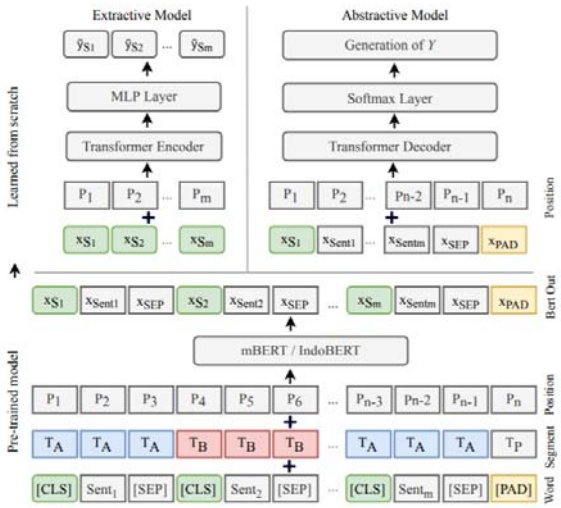
1.3 Tujuan Skripsi

Tujuan dari skripsi ini adalah membuat peringkasan otomatis secara ekstraktif pada berita berbahasa Indonesia dengan menggunakan *Bidirectional Encoder Representations from Transformers* sehingga dapat mengurangi waktu baca dan mendapatkan informasi yang relevan.

1.4 Ruang Lingkup

Ruang lingkup dibatasi pada:

1. Input yang akan digunakan berupa dataset berita berbahasa Indonesia dengan format json yang diambil dari Indosum yang merupakan hasil penelitian yang dilakukan oleh Kurniawan dan Louvan (Kurniawan & Louvan, 2018).
2. Output berupa kalimat-kalimat yang dinilai penting oleh metode dan diambil dari bacaan berita aslinya.
3. Hasil ringkasan dapat dimodifikasi panjangnya sesuai dengan persentase maksimum kalimat yang dapat diinput oleh pengguna.
4. *Pre-trained model* yang digunakan adalah indobert-base-uncased (Koto et al., 2020).
5. Evaluasi dari ringkasan yang dihasilkan akan menggunakan *ROUGE* dimana akan dibandingkan metode BERT dengan metode-metode lain pada dataset yang sama.
6. Menggunakan bahasa pemrograman php dan python.
7. Alur program yang akan dibuat mengikuti bagan berikut (mengikuti model ekstraktif):



Gambar 1.1 Arsitektur model BERT dalam menghasilkan ringkasan

Sumber: Koto, F., Lau, J. H., & Baldwin, T. (2020). *Liputan6: A Large-scale Indonesian Dataset for Text Summarization*. http://arxiv.org/abs/2011.00679

1.5 Metodologi Penelitian

Dalam melakukan penelitian, terdapat beberapa langkah yang dilakukan yaitu:

1. Studi Literatur
    1.1. Teori *News Summarization*
    1.2. Metode *Bidirectional Encoder Representations from Transformers*
    1.3. Arsitektur *Transformer*
    1.4. Evaluasi *ROUGE*
2. Pengambilan dataset
    2.1. Pengambilan dataset dari Indosum
    2.2. Analisa dataset untuk identifikasi pola dataset
3. Perencanaan dan Pembuatan Program
    3.1. Melakukan pembacaan dataset ke dalam sistem
    3.2. Melakukan *preprocessing* sederhana terhadap data
    3.3. Mengimplementasikan *BERT Summarization* ke dalam sistem
4. Pengujian dan Analisis Program
    4.1. Melakukan *testing* evaluasi ringkasan yang dihasilkan sistem dengan *ROUGE*
    4.2. Melakukan analisa metode *BERT* terhadap metode-metode lainnya
5. Pengambilan Kesimpulan
    5.1. Membuat kesimpulan mengenai hasil penelitian dari yang sudah dilakukan
    5.2. Membuat saran untuk penelitian serupa kedepannya

1.6 Sistematika Penulisan

Sistematika penulisan untuk penyusunan skripsi ini dibagi menjadi beberapa bab, yaitu:

BAB I: PENDAHULUAN

Bab 1, berisi penjelasan tentang latar belakang, perumusan masalah, tujuan skripsi, ruang lingkup, metodologi penelitian, sistematika penulisan mengenai penelitian yang dilakukan dalam skripsi yang dikerjakan.

BAB II: LANDASAN TEORI

Bab 2, berisi teori-teori yang menjelaskan mengenai dataset yang digunakan, *Bidirectional Encoder Representations from Transformers*, *Transformer*, dan *ROUGE score*.

BAB III: ANALISIS DAN DESAIN SISTEM

Bab 3, berisikan analisis dan perencanaan alur pembuatan keseluruhan sistem dari aplikasi yang dibuat.

BAB IV: IMPLEMENTASI SISTEM

Bab 4 berisikan tentang pembuatan aplikasi dan segmen-segmen program yang dibuat sesuai dengan desain sistem pada Bab 3.

BAB V: PENGUJIAN SISTEM

Bab 5 berisikan hasil dari pengujian implementasi aplikasi dan hasil pengujian yang telah dilakukan pada aplikasi.

BAB VI: KESIMPULAN DAN SARAN

Bab 6 berisikan kesimpulan dari penjelasan hasil pengujian sistem yang dicapai dan menjelaskan saran untuk pengembangan lebih lanjut.

2. LANDASAN TEORI

2.1 Tinjauan Pustaka

Subbab mengenai teori umum menjelaskan teori yang ada dalam penelitian kali ini secara umum yang dapat mendukung pemahaman dasar dari konsep dasar dalam penelitian ini.

2.1.1 News Summarization

Ringkasan merupakan suatu teks singkat yang dihasilkan dari kumpulan teks panjang namun tetap menyimpan informasi penting dari teks asalnya (Joshi et al., 2019). Informasi yang menunjukkan ide utama dapat diberikan penilaian dari seberapa tersambungnya informasi dengan tema dan judul berita serta penilaian dari pembaca. Lalu, untuk menentukan informasi penting maka lebih akurat bila dipilih oleh manusia. Karena itulah, perlu dasar untuk melakukan pelatihan terhadap ringkasan yang dibuat oleh manusia serta sudah dipercaya sebagai ringkasan yang baik dengan mengandung informasi penting dari dokumen.

Berdasarkan dari hasilnya, metode untuk peringkasan otomatis dapat dibedakan menjadi dua kategori yaitu abstraktif dan ekstraktif. Metode abstraktif memiliki tujuan untuk melakukan *paraphrase* dari informasi yang dipilih pada dokumen yang akan diringkas. Kemudian, untuk metode ekstraktif memiliki tujuan untuk membuat ringkasan dengan memilih kalimat maupun kata-kata penting dari dokumen yang akan diringkas. Dengan adanya ringkasan dapat mempersingkat waktu untuk menemukan informasi penting dari bacaan (El-Kassas et al., 2020).

2.1.2 Indosum

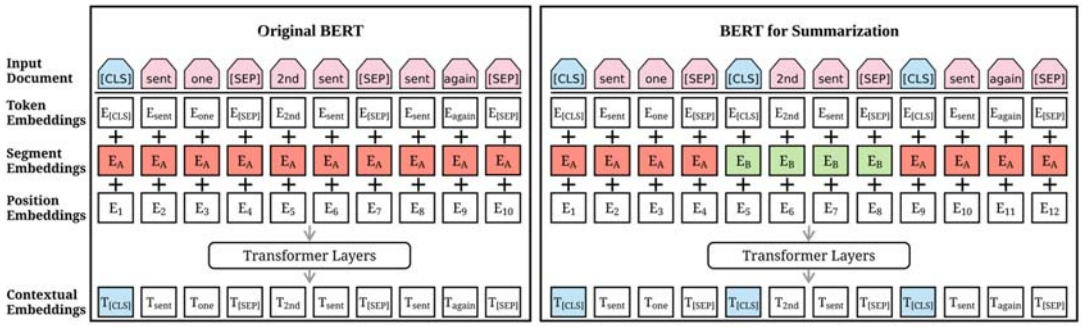
Indosum merupakan *dataset* berita berbahasa Indonesia yang merupakan hasil penelitian yang dilakukan oleh Kurniawan dan Louvan. *Dataset* pada penelitian dapat dijadikan sebagai tolok ukur dalam pembuatan ringkasan otomatis berbahasa Indonesia. *Dataset* ini berbentuk json. Selain itu, dataset ini juga berisi 18.774 artikel berita yang diambil dari berbagai portal berita seperti CNN Indonesia, Kumparan, dan portal berita lainnya. Dalam *dataset* ini terdapat *category* yang menunjukkan kategori berita, *gold label* yang merupakan label *gold summarization* untuk peringkasan ekstraktif, *id* unik tiap berita dari berita yang bersangkutan, *source* yang berisi sumber berita, *source url* yang berisi *url* dari berita, *paragraph* yang berisi berita yang sudah dilakukan pemecahan ke dalam *array*, dan *summary* yang berisi ringkasan berita yang dibuat oleh manusia dan bersifat abstraktif (Kurniawan & Louvan, 2018).

Berikut ini merupakan salah satu contoh isi data dari *dataset*:

{
"category": "olahraga",
"gold_labels": [[true], [false], … ,[false]],
"id": "1503595411-evaluasi-indonesia-untuk-hadapi-malaysia",
"paragraphs": [[["JUARA.NET", ",", "KUALA", "LUMPUR", "-", "Tim", "bulu", "tangkis", "putra", "Indonesia", "akan", "berhadapan", "dengan", "Malaysia", "pafa", "final", "beregu", "SEA", "Games", "2017", "di", "Axiata", "Arena", ",", "Bukit", "Jalil", ",", "Malaysia", ",", "Kamis", "(", "24", "/", "8", "/", "2017", ")", "."]], … ,[["\"", "Saya", "rasa", "pertandingan", "akan", "ramai", ",", "baik", "di", "tunggal", "maupun", "ganda", ",", "\"", "ujar", "Indra", "."]], [["Final", "beregu", "putra", "dijadwalkan", "pada", "Kamis", "(", "24", "/", "8", "/", "2017", ")", "mulai", "pukul", "12.45", "waktu", "setempat", "atau", "13.45", "WIB", "."]]],
"source":"juara.net",
"source_url":"http://juara.bolasport.com/read/raket/bulu-tangkis/1820702-evaluasi-indonesia-untuk-hadapi-malaysia-pada-final-beregu-putra-sea-games-2017",
"summary": [["Tim", "bulu", "tangkis", "putra", "Indonesia", "akan", "berhadapan", "dengan", "Malaysia", "pafa", "final", "beregu", "SEA", "Games", "2017", "di", "Axiata", "Arena", ",", "Bukit", "Jalil", ",", "Malaysia", ",", "Kamis", "(", "24", "/", "8", "/", "2017", ")", "."], …, ["Pemain", "juga", "sebaiknya", "tak", "terpengaruh", "apa", "pun", "dan", "fokus", "pada", "pertandingan", "."]]
}

2.1.3 Bidirectional Encoder Representations from Transformers (BERT)

BERT merupakan arsitektur *pre-trained model multi-layer bidirectional transformer encoder* untuk keperluan *Natural Language Processing*. Model ini dapat digunakan untuk kebutuhan seperti *text classification*, *question answering*, dan lain-lain (Clark et al., 2019). Mengenai struktur dari BERT dapat dilihat pada Gambar 2.1.

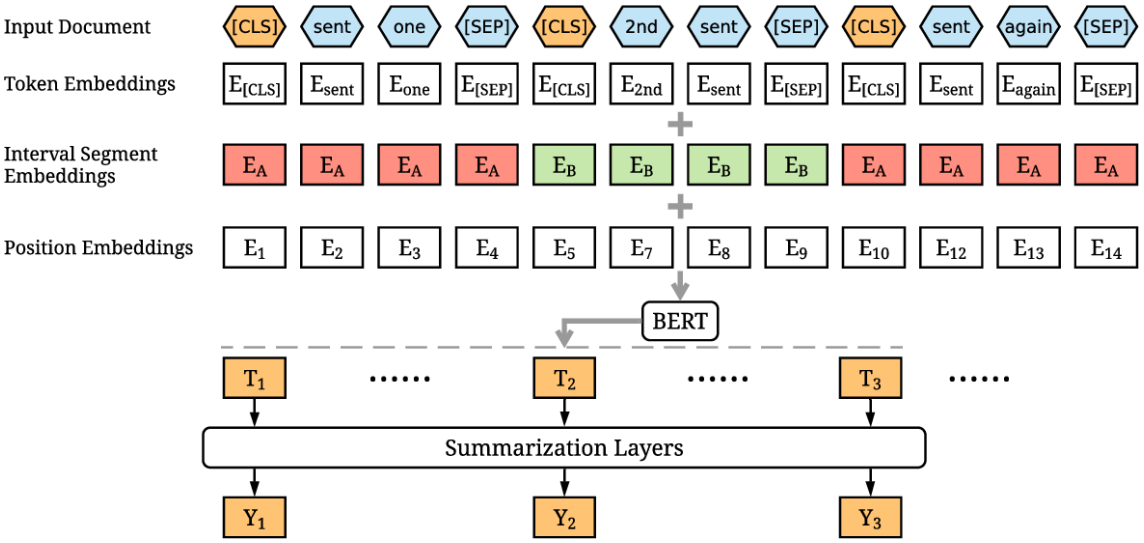


Gambar 2.1 Perbandingan struktur BERT

Sumber: Liu, Y., & Lapata, M. (2019). Text summarization with pretrained encoders. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3730–3740. https://doi.org/10.18653/v1/d19-1387

Pada BERT terdapat dua tahapan yang dilakukan yaitu *pre-training* dan *fine-tuning*. Selama *pre-training*, akan dilakukan dengan *Masked Language Models* (MLM) dimana secara *random* melakukan *mask* terhadap beberapa *token* dari input kalimat dan tujuannya untuk memprediksi kata yang di-*masked* berdasarkan pada konteks kalimat tersebut. Pada BERT, input dari suatu teks kalimat akan diubah menjadi *token sequence*. Dimana terdapat *token* [CLS] yang melambangkan mulainya suatu kalimat dan *token* [SEP] yang merupakan *token* untuk memisahkan antar kalimat. Selain itu, juga terdapat *token* [PAD] untuk melakukan penambahan *padding* atau *token* kosong ke dalam kalimat untuk memenuhi *fixed length* dari suatu kalimat yang diinputkan. *Fixed length* pada model BERT adalah 512. Untuk melakukan pelatihan terhadap representasi *bidirectional* maka dilakukan *mask* dengan persentase tertentu dari input *token* secara *random* dan dilanjutkan dengan memprediksi *masked token* yang dilatih pada suatu *pre-training task* dengan *Next Sentence Prediction* (NSP). Lalu, untuk *fine-tuning* model BERT akan melakukan inisialisasi parameter yang sudah dilatih sebelumnya (Devlin et al., 2019). Pada Gambar 2.1 menunjukkan bahwa terdapat perbedaan untuk implementasi model BERT dimana untuk penerapan dalam pembuatan ringkasan dilakukan beberapa perubahan dari model asli BERT. Perubahan terletak pada token [CLS] dan [SEP] yang disisipkan pada setiap awal dan akhir kalimat. Kemudian, saat input dokumen melebihi *fixed length* dari BERT maka akan dilakukan pemotongan terhadap kalimat yang berikutnya. Selain itu, pemberian *segment embedding* dilakukan secara selang-seling pada tiap kalimat. Hal tersebut dilakukan supaya model BERT dapat membedakan antar kalimat. *Position embedding* digunakan untuk

memberikan informasi posisi pada token kata (Liu & Lapata, 2019). Mengenai detail struktur lengkap implementasi dari model BERT untuk pembuatan ringkasan dapat dilihat pada Gambar 2.2.



Gambar 2.2 Detail lengkap untuk struktur BERT Summarization

Sumber: Liu, Y. (2019). Fine-tune BERT for Extractive Summarization. *ArXiv*. http://arxiv.org/abs/1903.10318

2.1.3.1 Token Embeddings

Pada *token embeddings* merupakan proses awal bagi suatu input teks dokumen. Proses yang akan dilakukan adalah mengubah input kata menjadi representasi vektor dengan dimensi yang sudah ditetapkan. Pada BERT, tiap kata akan direpresentasikan sebagai vektor *768-dimensional*. Terdapat 2 tahapan yang dilakukan pada *token embeddings*. Tahap pertama, suatu input teks akan dilakukan proses tokenisasi yaitu memecah kalimat menjadi kata yang kemudian juga disisipkan dengan token [CLS] pada tiap awal kalimat dan [SEP] pada tiap akhir kalimat yang sudah ditokenisasi. Tujuan dari adanya token [CLS] adalah mengumpulkan fitur untuk representasi dari kalimat yang mengikutinya dan [SEP] untuk keperluan pemisahan antar kalimat. Tokenisasi dilakukan dengan menggunakan metode *WordPiece tokenization*. *WordPiece tokenization* merupakan suatu metode yang bertujuan untuk memperoleh keseimbangan antara ukuran kata *vocab* dan *out-of-vocab*. Penggunaan dari *WordPiece tokenization* memampukan BERT untuk menyimpan 30552 kata dalam *vocab* nya. Tahap kedua,

akan dilakukan pengubahan *WordPiece Token* menjadi representasi vektor 768-*dimensional* (Devlin et al., 2019).

Pada penerapan dari *WordPiece tokenization* terdapat karakter spesial ## berfungsi sebagai penanda untuk setiap kata yang disegmentasi kecuali untuk *subword* pertama tidak menggunakan penanda ## (Devlin et al., 2019). Bila terdapat kalimat 'saya sedang memakan nasi goreng dengan mentimun' maka akan diubah menjadi ['saya', 'sedang', 'memakan','nasi','goreng','dengan','ment','##imu','##n']. Bila suatu kata tidak dikenali dalam *dictionary* yang mengubah string menjadi id maka akan dilakukan pemecahan kata menjadi sub-kata dengan sub-kata pertama tanpa diikuti penanda ##.

2.1.3.2 Interval Segment Embeddings

Pada *interval segment embeddings* dilakukan untuk membedakan antar kalimat dengan menggunakan representasi vektor segmen *even/odd* yaitu dengan menyisipkan pelabelan secara selang-seling berdasarkan urutan kalimat terhadap *word embeddings*. Sebagai contoh terdapat input kalimat 'Saya suka memakan ikan goreng. Adik suka memakan ayam goreng.' maka kedua kalimat tersebut akan dilakukan tokenisasi. Setelah itu, dilakukan pelabelan terhadap kalimat tersebut dengan menggunakan 0 untuk kalimat urutan *i* genap dan 1 untuk kalimat urutan *i* ganjil. Pelabelan tersebut merupakan representasi vektor dari segmen kalimat (Devlin et al., 2019).

2.1.3.3 Position Embeddings

Position embeddings atau *position encoding* diperlukan sehingga model BERT dapat mengetahui posisi suatu teks dalam dokumen. Bila terdapat input teks seperti 'walaupun saya sedang lelah, saya tetap bekerja'. Dalam hal ini kata 'saya' pada 'saya sedang lelah' akan memiliki representasi vektor yang berbeda dengan kata 'saya' pada 'saya tetap bekerja' karena memiliki posisi yang berbeda. Tujuan dari *position embeddings* adalah untuk memberikan informasi mengenai posisi absolut dari tiap *token* karena pada struktur *transformer* tidak melakukan *encode* dengan sifat *sequential*.

Persamaan yang digunakan untuk membuat *position embeddings* pada *time step* genap:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \quad (2.1)$$

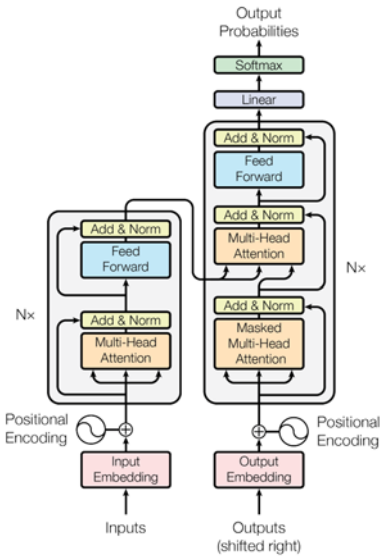
Persamaan yang digunakan untuk membuat *position embeddings* pada *time step* ganjil:

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \quad (2.2)$$

Untuk kedua persamaan diatas, *pos* melambangkan posisi dan *i* merupakan dimensi. Tiap dimensi dari *position embeddings* tersusun dari *sinusoidal position*. Model ini dipilih karena akan mempermudah dalam belajar untuk fokus terhadap posisi kalimat (Vaswani et al., 2017).

2.1.4 Transformer

Pada *transformer* terdapat beberapa komponen yaitu *attention* dan *feed forward network*. *Attention* pada *transformer* terdiri atas banyak *head* yang mempunyai *weight* berbeda-beda dan dapat menempati urutan posisi berbeda-beda (Vaswani et al., 2017). Berikut merupakan struktur dari *transformer*.



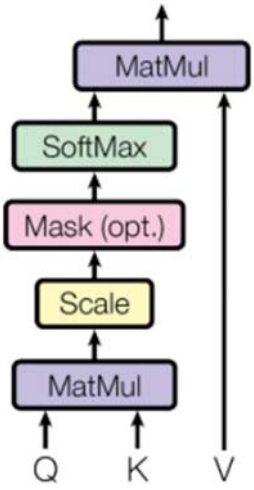
Gambar 2.3 Struktur *transformer*

Sumber: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *2017-Decem*(Nips), 5999–6009.

Pada *transformer* terdapat input *embeddings* yang mengubah kata menjadi *list* angka yang merepresentasikan kata yang diinputkan. Lalu, dilakukan *positional encoding* untuk menambahkan informasi mengenai posisi ke *input embeddings* dimana untuk *time step* ganjil akan dibuat dengan fungsi *cos* dan untuk *time step* genap akan dibuat dengan fungsi *sin* seperti penjelasan pada subbab 2.1.3.3. Pada *transformer* tersusun atas *sub-layer* yaitu *multi-head attention* dan *feed forward network*. Lalu, juga terdapat koneksi residual dan layer normalisasi dimana koneksi residual membantu supaya tidak kehilangan informasi dari input sebelumnya sedangkan layer normalisasi untuk menstabilkan nilai ke layer berikutnya sehingga tetap konsisten (Xiong et al., 2020). *Self-attention* dalam model *transformer* menerima *embedding* yang diperoleh dari perkalian antara matriks *query*, *key*, dan *value*. Detail dari proses kalkulasi tersebut dapat dilihat pada persamaan 2.3. Mekanisme *self-attention* digunakan pada model untuk melakukan asosiasi tiap kata dari input ke kata lain (Vaswani et al., 2017).

Self-attention diperoleh dengan menggunakan persamaan *scaled-dot product attention* berikut:

$$Attention\ (Q, K, V) = \ softmax_k\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$



Gambar 2.4 Diagram *scaled dot product attention*

Sumber: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *2017-Decem*(Nips), 5999–6009.

Dimana pada persamaan (*2.3*) akan dilakukan penghitungan skor *attention* menggunakan *Query* (Q), *Key* (K), dan *Value* (V). Tahap awal, akan dilakukan perkalian titik dari matriks *query* dengan matriks *key* dari masing-masing kata. Kemudian, diulangi dengan perkalian titik dari matriks *query* dengan matriks *key* dari kata berikutnya dan seterusnya. Dengan melakukan perkalian akan menghasilkan matriks skor. Skor digunakan untuk mengukur seberapa banyak fokus untuk kata-kata dari urutan input dengan kata pada posisi tertentu. Jadi setiap kata akan memiliki skor yang sesuai dengan kata lain dalam langkah waktu. Semakin tinggi skor, maka akan semakin fokus. Kemudian, akan dilakukan penskalaaan nilai untuk mendapatkan gradien yang lebih stabil dimana akan dilakukan pembagian matriks skor dengan akar kuadrat dari dimensi *key* (d_k). Penskalaan dilakukan sehingga nilai dari hasil perkalian *dot* antara matriks *query* dan *key* tidak menjadi terlalu besar. Matriks berskala tersebut kemudian dilewatkan melalui *softmax* untuk mendapatkan bobot perhatian. Hasil dari *softmax* akan menghasilkan nilai diantara 0 dan 1. Dengan melakukan *softmax*, skor yang lebih tinggi semakin diperkuat dan skor yang lebih rendah akan diperkecil sehingga memungkinkan model untuk lebih memperhatikan tentang kata-kata mana yang akan dipilih. Setelah memperoleh keluaran *softmax*, matriks ini dikalikan dengan matriks *value*. Skor *softmax* yang lebih tinggi akan membuat nilai kata yang dipelajari model menjadi lebih penting sedangkan skor yang rendah akan menunjukkan kata-kata yang tidak relevan (Vaswani et al., 2017). Dalam melakukan klasifikasi terhadap informasi yang diambil akan digunakan *linear layer* untuk *feed forward neural network* dan fungsi *aktivasi sigmoid* (Deng & Liu, 2018).

2.1.5 Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

ROUGE merupakan salah satu cara untuk melakukan evaluasi kualitas dari suatu ringkasan yang dibuat secara otomatis. Ringkasan akan dibandingkan dengan *gold summary* yang ada. *Gold summary* merupakan ringkasan dibuat oleh manusia dan menjadi standar bagi teks yang akan diringkas. Pemilihan *ROUGE* dalam evaluasi ringkasan dikarenakan sudah menjadi standar evaluasi dan telah banyak digunakan oleh penelitian lain. Terdapat 2 jenis *ROUGE* yang digunakan dalam melakukan peringkasan yaitu *ROUGE-N* dan *ROUGE-L*. *ROUGE-N* merupakan pencarian nilai *recall* berdasarkan *n-gram* sedangkan *ROUGE-L* menggunakan *longest common subsequence* (Lin, 2004). *ROUGE-N* digunakan untuk menilai dari sisi *informativeness* atau seberapa banyak kesamaan informasi yang terkandung dalam hasil ringkasan sistem dengan referensi ringkasan. *ROUGE-L* akan menilai dari sisi *fluency* atau

seberapa lancar hasil ringkasan yang dihasilkan dengan melihat urutan kesamaan kata dalam suatu kalimat terhadap referensi ringkasan (Lin, 2004).

Persamaan yang digunakan untuk menghitung *ROUGE-N*:

$$\frac{\sum_{S\epsilon\{ReferenceSummaries\}}\sum_{gram_n\,\epsilon\,S} Count_{match}(gram_n)}{\sum_{S\epsilon\{ReferenceSummaries\}}\sum_{gram_n\,\epsilon\,S} Count(gram_n)} \quad (2.4)$$

Dimana *n* merupakan panjang dari *n-gram* dan $count_{match}(gram_n)$ merupakan jumlah *n-gram* yang cocok pada ringkasan yang dibuat otomatis dengan yang dijadikan referensi. Lalu, $count(gram_n)$ merupakan jumlah total dari *n-gram* yang terjadi dalam referensi ringkasan.

Persamaan yang digunakan untuk menghitung *ROUGE-L*:

$$R_{lcs} = \frac{LCS\ (X,Y)}{m} \quad (2.5)$$

$$P_{lcs} = \frac{LCS\ (X,Y)}{n} \quad (2.6)$$

$$F_{lcs} = \frac{(1+\ \beta^2)R_{lcs}P_{lcs}}{R_{lcs} +\ \beta^2\ P_{lcs}} \quad (2.7)$$

Dimana untuk perhitungan *ROUGE-L* terdapat pada F_{lcs}. Untuk R_{lcs} dan P_{lcs} didapatkan dengan menghitung kesamaan antara kalimat ringkasan referensi X dan kalimat ringkasan sistem Y dimana untuk m merupakan panjang kalimat ringkasan referensi X dan n merupakan panjang kalimat ringkasan sistem Y. Lalu untuk nilai β didapatkan dari P_{lcs}/R_{lcs}. Lalu, untuk nilai *ROUGE-L* didapat lewat perhitungan F_{lcs}.

2.2 Tinjauan Studi

Subbab mengenai penelitian-penelitian yang serupa yang sudah pernah dilakukan sebelum penelitian ini, hal hal yang diangkat seperti masalah yang diangkat dalam penelitian sebelumnya, metode yang digunakan dalam penelitian sebelumnya, hasil yang diperoleh dari penelitian sebelumnya, dan perbedaan penelitian sebelumnya dengan penelitian yang akan dilakukan.

2.2.1 Neural Summarization by Extracting Sentences and Words (Cheng & Lapata, 2016)

Masalah yang diangkat oleh peneliti adalah besarnya jumlah data teks yang ada sehingga terdapat kebutuhan dengan tujuan untuk mempermudah pembacaan dengan mengembangkan peringkasan otomatis dalam membuat ringkasan yang lebih pendek dari dokumen dengan tetap menjaga makna informasi dari dokumen yang diringkas.

Metode yang digunakan pada penelitian ini adalah *neural network* berbasis *encoder* dan *attention-based content extractor* dimana *encoder* berperan untuk mendapatkan makna dari dokumen berdasarkan dari kalimat dan kata pada dokumen tersebut. Kemudian, *attention* akan memilih kalimat atau kata untuk menghasilkan ringkasan ekstraktif.

Hasil dari penelitian ini didapatkan bahwa penerapan dari metode yang diusulkan dalam menghasilkan ringkasan ekstraktif dengan pengukuran *ROUGE* adalah 47.4% untuk *ROUGE-1*, 23% untuk *ROUGE-2*, 43.5% untuk *ROUGE-L* pada dataset DUC 2002 serta 21.2% untuk *ROUGE-1*, 8.3% untuk *ROUGE-2*, dan 12% untuk *ROUGE-L* pada dataset DailyMail.

Perbedaan dari penelitian yang dilakukan oleh peneliti dengan penelitian yang akan dilakukan adalah pada penelitian ini menggunakan dataset berita berbahasa Inggris sedangkan pada penelitian yang akan dilakukan melakukan peringkasan otomatis pada berita berbahasa Indonesia dengan menggunakan dataset penelitian yaitu Indosum.

2.2.2 Peringkasan Ekstraktif Teks Bahasa Indonesia dengan Pendekatan Unsupervised Menggunakan Metode Clustering (Ismi & Ardianto, 2019)

Masalah yang diangkat oleh peneliti adalah volume informasi yang sangat banyak karena perkembangan teknologi informasi yang pesat sehingga harus memilah informasi yang penting untuk merangkum informasi tersebut. Proses meringkas informasi yang banyak juga memerlukan waktu yang lama untuk menyusuri setiap informasi dan mengambil informasi utama.

Metode yang diusulkan oleh peneliti adalah menggunakan metode *K-Means clustering* dalam membuat ringkasan ekstraktif secara otomatis. Sebelumnya akan dilakukan *text preprocessing* dengan *tokenization* (pemisahan dokumen teks untuk mendapatkan frasa dengan inti yang berbobot), *stopword removal* (menghilangkan kata-kata yang tidak memiliki makna), *stemming* (mengidentifikasi bentuk akar kata dengan menghapus imbuhan). Lalu, dilanjutkan dengan pembobotan TF-IDF (bobot suatu istilah akan semakin besar bila sering muncul pada suatu dokumen dan semakin kecil bila muncul dalam banyak dokumen). Kemudian, diterapkan

K-Means clustering dimana setiap klaster dapat terdiri dari beberapa kalimat dan tiap klaster akan mewakili satu kalimat. Satu kalimat tersebut nantinya akan menyusun ringkasan teks secara keseluruhan untuk menghasilkan ringkasan ekstraktif.

Hasil dari penelitian menunjukkan bahwa dengan penerapan *K-Means Clustering* dalam sistem peringkasan ekstraktif otomatis diperoleh nilai *F1-Score* dari *ROUGE-1* sebesar 49.37%, *ROUGE-2* sebesar 38.18% dan *ROUGE-L* sebesar 46.87%.

Perbedaan dari penelitian yang dilakukan oleh peneliti dengan penelitian yang akan dilakukan adalah pada penelitian ini menggunakan metode *clustering* dalam membuat ringkasan berita berbahasa Indonesia sedangkan pada penelitian yang akan dilakukan menggunakan metode BERT dalam menghasilkan ringkasan berita berbahasa Indonesia secara otomatis.

2.2.3 Penerapan Recurrent Neural Network untuk Pembuatan Ringkasan Ekstraktif Otomatis pada Berita Berbahasa Indonesia (Halim et al., 2020)

Masalah yang diangkat oleh peneliti adalah perkembangan internet yang sangat pesat sehingga membuat kewalahan dalam mengikuti informasi yang tersedia secara online. Ringkasan dapat dibuat manual oleh manusia akan tetapi memerlukan waktu yang cukup lama karena informasi yang tersedia sangat banyak. Berita berbahasa Indonesia dipilih karena belum terlalu berkembang bila dibandingkan dengan berita berbahasa Inggris. Kemudian, untuk tingkat kualitas berita berbahasa Indonesia masih cukup rendah dimana kebanyakan masih dibawah 50%.

Metode yang diusulkan oleh peneliti adalah menggunakan metode *Recurrent Neural Network* untuk membuat ringkasan berita berbahasa Indonesia secara otomatis dengan menggunakan dataset Indosum.

Hasil pada penelitian ini adalah diperoleh *F1-Score* sebesar 57.01 untuk *ROUGE-1*, 51.17 untuk *ROUGE-2* dan 55.10 untuk *ROUGE-L* dengan referensi abstraktif serta *F1-Score* sebesar 84.10 untuk *ROUGE-1*, 83.10 untuk *ROUGE-2*, dan 83.31 untuk *ROUGE-L* dengan referensi ekstraktif.

Perbedaan dari penelitian yang dilakukan oleh peneliti dengan penelitian yang akan dilakukan adalah pada penelitian ini menggunakan *Recurrent Neural Network* sedangkan pada penelitian yang akan dilakukan menggunakan metode BERT.

2.2.4 Arabic Text Summarization Using AraBERT Model Using Extractive Text Summarization Approach (Nada et al., 2020)

Masalah yang diangkat oleh peneliti adalah informasi teks yang tersedia secara *online* semakin berkembang dengan pesat sehingga jumlah informasi semakin banyak dan mengakibatkan pembaca kesulitan dalam membaca kumpulan teks tersebut. Selain itu, sebagian besar penelitian masih dilakukan pada teks berbahasa Inggris. Peneliti menyebutkan bahwa pada penelitian teks berbahasa Arab masih rendah. Beberapa permasalahan lainnya adalah kompleksitas dari bahasa Arab dan *library* NLP untuk bahasa Arab masih kurang mendukung.

Metode yang diusulkan oleh peneliti adalah menggunakan metode AraBERT dimana pada input awalnya dilakukan operasi *preprocessing* untuk menghilangkan kalimat yang tidak memiliki makna. Kemudian, dilakukan pengubahan teks menjadi *sentence embedding* sebelum diteruskan pada model AraBERT. Setelah itu, akan dilakukan proses *clustering* dengan menggunakan *K-Means* pada hasil matriks *embedding* dari model AraBERT. Lalu, dari hasil *clustering* akan dipilih kalimat terdekat dengan tiap *cluster centroid* sebagai hasil ringkasan.

Hasil dari penelitian yang telah dilakukan dengan metode yang diusulkan yaitu diperoleh *F-measure* sebesar 0.54 untuk ROUGE-1 dan *F-measure* sebesar 0.51 untuk ROUGE-2 dengan menggunakan pendekatan ekstraktif.

Perbedaan dari penelitian yang dilakukan oleh peneliti dengan penelitian yang akan dilakukan adalah pada penelitian ini dilakukan pada teks berita berbahasa Arab sedangkan pada penelitian yang akan dilakukan menggunakan model BERT pada teks berita berbahasa Indonesia.

3. ANALISIS DAN DESAIN SISTEM

Pada bab ini akan dibahas analisa data yang dilakukan dan desain sistem yang dibuat beserta dengan gambaran alur sistem.

3.1 Analisa Masalah

Pada subbab ini akan dibahas mengenai analisa masalah mengenai ringkasan ekstraktif dan metode yang digunakan pada penelitian ini. Berita merupakan salah satu sumber informasi yang tersedia secara online, akan tetapi jumlah informasi yang tersedia pada sumber berita online sangat banyak sehingga membuat manusia kerepotan untuk mengikutinya satu persatu. Selain itu, membaca keseluruhan informasi berita terkadang juga memerlukan waktu yang lama sehingga perlu dilakukan pembuatan ringkasan dari berita-berita yang tersedia tersebut. Ringkasan ekstraktif digunakan pada penelitian ini karena akan mengambil dari teks aslinya sehingga diharapkan informasi yang dihasilkan dalam ringkasan tidak berubah terlalu jauh dari topik yang ada dalam berita asli. Kemudian, pada penelitian ini akan menghasilkan ringkasan berita dari berita asli karena untuk proses pembuatan ringkasan dengan mengambil langsung dari berita asli dapat membantu mengatasi beberapa masalah seperti mengurangi waktu baca dengan mengambil kalimat penting dari berita asli dan membantu mengurangi beban kerja manusia dalam membaca informasi berita dengan menghilangkan informasi yang kurang relevan. Penerapan metode dari BERT pada ringkasan ekstraktif dilakukan untuk mengatasi keterbatasan RNN dalam memproses kata demi kata yang dilakukan secara berurutan dimana pada metode BERT akan langsung memproses seluruh input kata secara bersamaan dalam melakukan proses pelatihan model sehingga diharapkan melalui penelitian ini model dapat belajar lebih baik dengan menggunakan metode BERT. Selain itu, pada penelitian yang telah dilakukan oleh Liu dan Lapata, metode BERT yang digunakan dapat memperoleh hasil yang baik saat diterapkan pada berita berbahasa Inggris (Liu & Lapata, 2019).

3.2 Analisa Data

Pada subbab ini akan dibahas mengenai analisa dari dataset yang telah diambil dan pengolahan yang akan direncanakan.

3.2.1 Pengambilan Data

Data yang akan diambil dari dataset terdapat 3 bagian yaitu ‘gold_labels’, ‘paragraphs’, dan ‘summary’. Dataset berita berbahasa Indonesia yang digunakan diambil dari penelitian yang telah disediakan oleh Kurniawan dan Louvan. Dataset memiliki format *json* dan dibagi menjadi 3 bagian yaitu untuk *train*, *test*, dan *dev*. Selain itu, dataset juga dibagi menjadi 5 *fold* dimana tiap *fold* akan memiliki data sebanyak 18.774 berita. Dataset *train* digunakan untuk proses *training*, *test* untuk proses *test*, dan *dev* untuk proses evaluasi *training*.

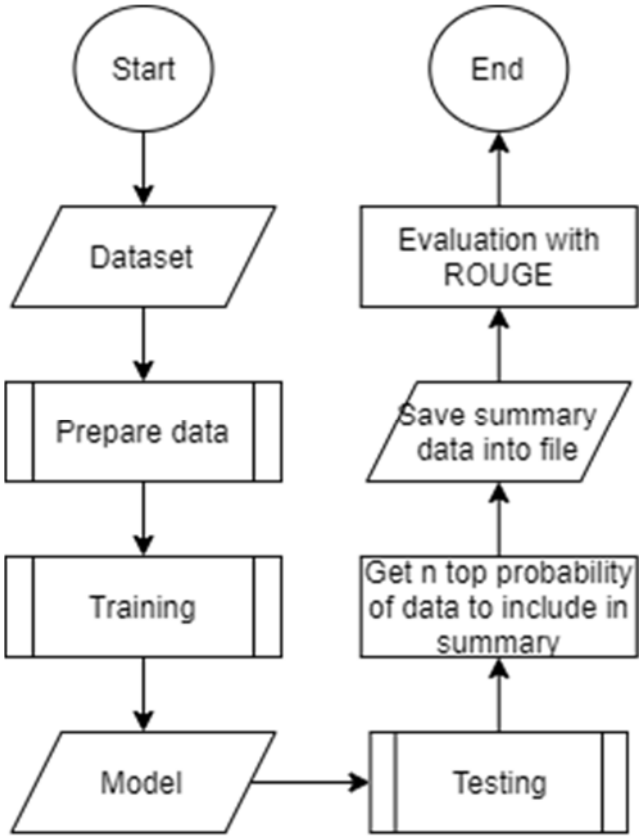
3.2.2 Pengolahan Data

Sebelum data digunakan ke dalam program, maka data akan diolah terlebih dahulu sehingga memudahkan pada implementasi ke dalam program. Beberapa hal yang akan dilakukan dalam pengolahan data diantaranya:

1. Menggunakan data yang terdapat pada bagian ‘gold_labels’, ‘paragraphs’, dan ‘summary’ dari dataset. ‘gold_labels’ digunakan untuk proses *training* dan pengujian dengan referensi ekstraktif, ‘paragraphs’ untuk isi berita yang akan diringkas, dan ‘summary’ untuk melakukan pengujian dengan referensi abstraktif.
2. Membuka array pada bagian ‘paragraphs’ untuk disusun menjadi satu kesatuan dokumen berita dan tiap kalimat dipisahkan dengan *delimiter* ‘<q>’.
3. Membuka array pada bagian ‘gold_labels’ untuk menunjukkan kalimat yang dipilih sebagai ringkasan ekstraktif yang kemudian diubah menjadi ‘labels’ dan tiap label dipisahkan dengan *delimiter* ‘<q>’.
4. Membuka array pada bagian ‘summary’ untuk diubah menjadi ‘target’ yang digunakan sebagai target dari referensi abstraktif yang akan diuji dan tiap kalimat dipisahkan dengan *delimiter* ‘<q>’.

3.3 Desain Sistem

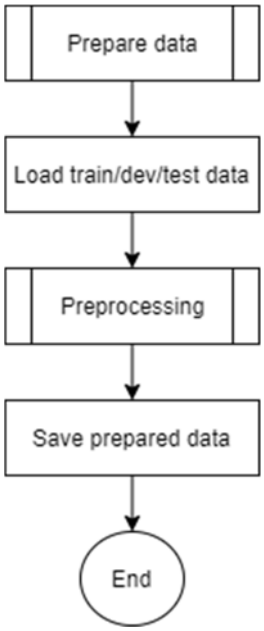
Dalam melakukan peringkasan secara ekstraktif, gambaran keseluruhan sistem yang akan diterapkan pada penelitian ini dapat dilihat pada Gambar 3.1. *Library* yang akan digunakan adalah *pytorch* dan *pytorch-lightning* untuk penerapan *encoder transformer* dan *feed forward neural network*. Kemudian, untuk model BERT akan menggunakan *library huggingface* (Wolf et al., 2020).



Gambar 3.1 Alur sistem secara keseluruhan

3.3.1 BERT Embedding

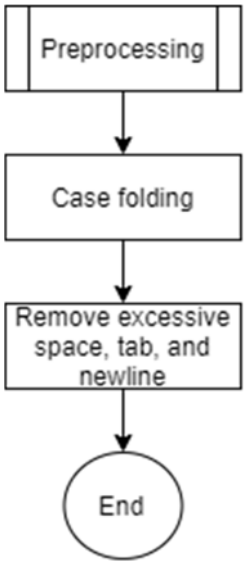
Dalam penerapan dari BERT akan memperhatikan konteks dari kalimat dimana *embedding* yang dihasilkan bisa lebih dari satu tergantung dari konteks dari kalimat (*context dependent*) sedangkan untuk model yang menggunakan *word2vec* akan menghasilkan representasi vektor yang sama untuk kata yang sama atau dengan kata lain sudah ditetapkan pada satu nilai vektor tertentu (*context independent*). Dalam penerapan BERT *embedding* akan menggunakan model indobert-base-uncased yang diambil dari *library huggingface* yang telah dilakukan training sebelumnya pada *wikipedia*, korpus web, dan artikel berita berbahasa Indonesia dengan total sejumlah 220 juta kata (Koto et al., 2020). Selain itu, juga akan digunakan bert-base-multilingual-uncased sebagai perbandingan dengan indobert-base-uncased. Sebelum dilakukan training pada model BERT akan disiapkan terlebih dahulu datanya dengan melakukan persiapan data yang dapat dilihat pada Gambar 3.2.



Gambar 3.2 Alur persiapan data

3.3.2 Preprocessing

Preprocessing yang akan dilakukan pada penelitian ini adalah hanya dengan mengubah data teks menjadi huruf kecil. Kemudian, dilanjutkan dengan merapikan data dari karakter spasi, *tab*, dan *newline* yang berlebihan. *Preprocessing* ini dilakukan untuk membantu model dalam mempelajari data. Selain itu, *preprocessing* seperti *stopword removal* dan *stemming* tidak dilakukan pada penelitian ini dengan alasan supaya model BERT dapat menangkap dengan lengkap hubungan dari tiap kalimat. Alur *preprocessing* dapat dilihat pada Gambar 3.3.



Gambar 3.3 Alur *preprocessing*

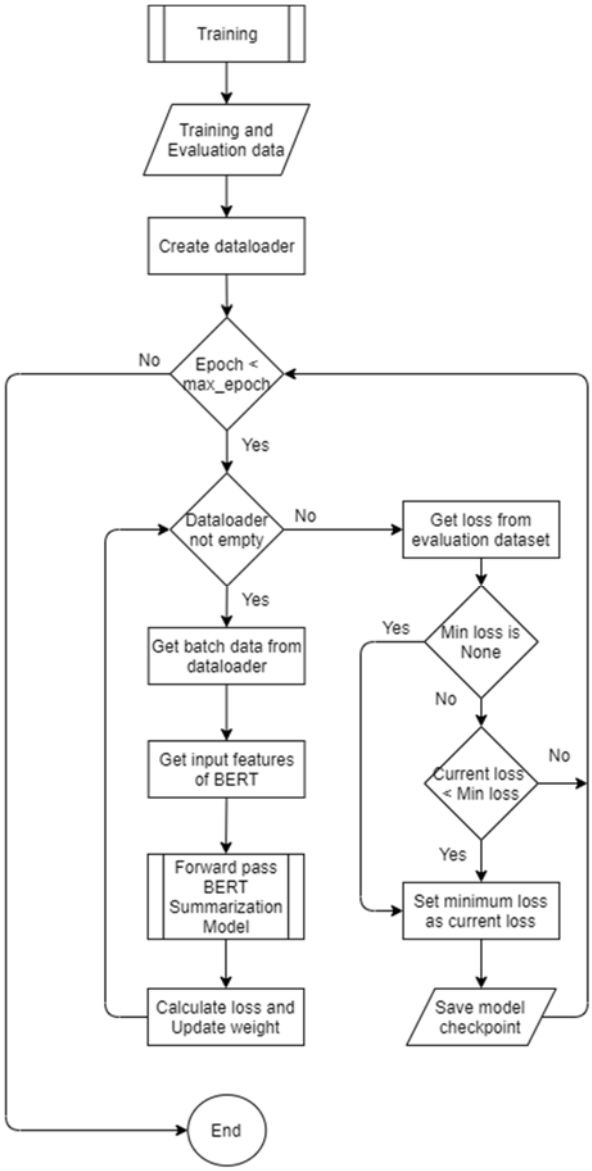
3.3.3 Training

Proses training dimulai dengan input model BERT yang digunakan dan dataset sebagai training dan evaluasi. Kemudian, akan dilakukan inisialisasi parameter *transformer encoder*, *optimizer*, dan lain-lain. Optimizer yang digunakan adalah *AdamW optimizer* dan fungsi *loss* yang akan digunakan adalah *binary cross entropy*. *AdamW optimizer* dipilih karena memiliki performa yang baik dengan melakukan modifikasi terhadap *weight decay*. Setelah itu, *training* akan dilakukan dalam beberapa *epoch*. Dalam satu *epoch*, proses akan dijalankan beberapa kali bergantung pada jumlah *batch*.

Pencarian *loss* menggunakan fungsi *loss* dengan target label sebagai ukuran kebenaran. Selama melakukan proses training, bila suatu *epoch* terpenuhi maka akan dilakukan proses evaluasi dimana model yang telah dilatih pada semakin baik atau tidak. Ketika dilakukan evaluasi model semakin baik maka model akan disimpan. Baik tidaknya suatu ditentukan berdasarkan dari *loss* yang didapat dari proses evaluasi dimana *loss* kecil menunjukkan bahwa model semakin membaik.

Tambahan susunan *transformer encoder* yang akan digunakan akan mengikuti penelitian yang telah dilakukan oleh Liu dan Lapata (Liu & Lapata, 2019), dimana model yang digunakan untuk peringkasan adalah *transformer encoder* untuk mengetahui relasi antar kalimat. Pada model yang digunakan akan mengambil pada level kalimat untuk diteruskan ke dalam layer klasifikasi (Liu, 2019). Layer klasifikasi akan dihitung berdasarkan dari *salience* suatu kalimat yaitu seberapa penting kalimat dari artikel berita yang bersangkutan. Setelah itu, akan dilakukan penghitungan dengan menggunakan fungsi aktivasi *sigmoid* untuk menghasilkan *array* yang menyimpan probabilitas dari tiap kalimat untuk masuk ke dalam ringkasan.

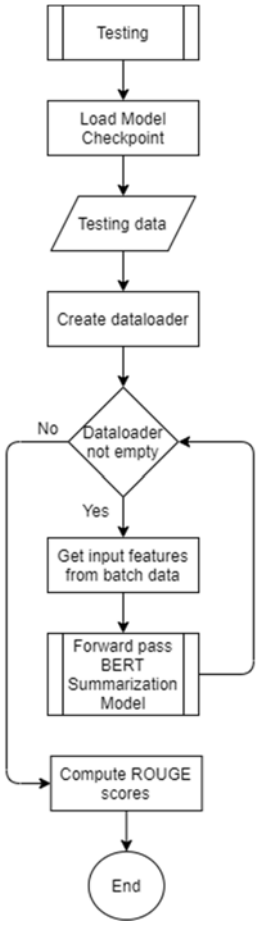
Proses evaluasi akan mirip dengan training hanya saja model tidak dilakukan training, namun langsung digunakan untuk mencari *loss* dari dataset evaluasi. Semua *loss* dari dataset akan digunakan untuk menentukan apakah model semakin baik atau tidak. Proses evaluasi bertujuan untuk menghindari terjadinya *overfitting* pada dataset *training*.



Gambar 3.4 Alur *training* secara garis besar

3.3.4 Testing

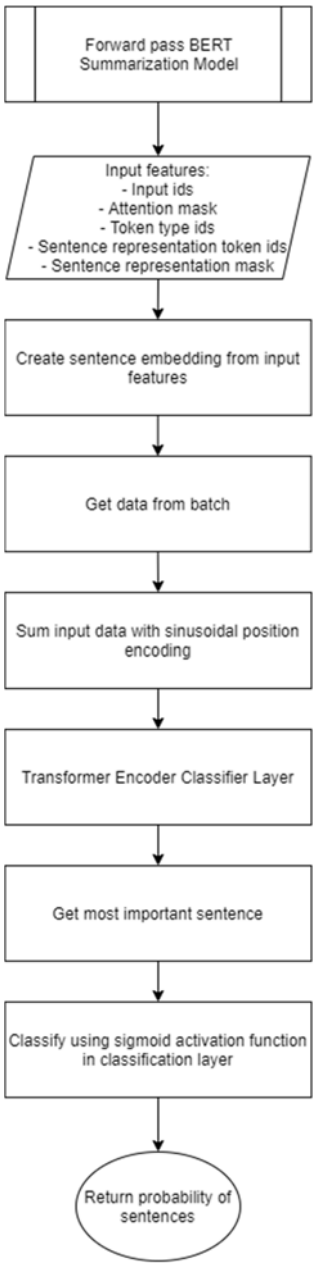
Pada awal proses testing akan dilakukan *load* dari *dataset test*. Lalu, akan dilanjutkan dengan melakukan *load* dari model BERT. Setelah itu, tiap data berita akan diproses oleh model untuk menghasilkan *array* berisi probabilitas dari tiap kalimat. Kemudian, data berita yang telah didapatkan probabilitasnya akan diambil sejumlah 'n' terbaik untuk dijadikan sebagai ringkasan. Hasil ringkasan akan disimpan ke dalam file teks yang selanjutnya dapat dilakukan penilaian dengan *ROUGE*.



Gambar 3.5 Alur *testing* secara garis besar

3.3.5 Forward Pass BERT Summarization Model

Pada penerapan dari *forward pass*, akan dilakukan pemilihan kalimat dari data teks berita pada *batch* tertentu dengan mempertimbangkan seberapa penting suatu kalimat berdasarkan artikel berita bersangkutan. Parameter tersebut kemudian akan dihitung dengan menggunakan fungsi *sigmoid* yang akan menghasilkan suatu array probabilitas dari tiap kalimat yang dapat masuk ke dalam ringkasan.



Gambar 3.6 Alur proses *forward pass BERT Summarization Model*

3.3.6 Evaluasi ROUGE

Dalam melakukan evaluasi terhadap ringkasan akan menggunakan ROUGE score. Penerapan dari ROUGE akan dibantu dengan menggunakan *library pyrouge*. Mengenai penjelasan dari ROUGE dapat dilihat pada subbab 2.1.

3.4 Desain Program

Pada subbab ini dibahas mengenai desain program yang akan dibuat berupa *website* secara *local*. Desain dari tampilan *website* dapat dilihat pada Gambar 3.7.

Gambar 3.7 Desain tampilan *website*

Untuk membuat ringkasan, maka *user* dapat menginputkan *url* dari sebuah portal berita online atau memilih sebuah file berita yang ingin diringkas. Kemudian, *user* dapat menginputkan persentase atau jumlah kalimat yang ingin diringkas. Persentase kalimat yang diisikan akan melakukan peringkasan berita sesuai dengan jumlah persentase tersebut.

4. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas tentang implementasi dari desain sistem yang sudah dijelaskan dalam Bab 3 Analisis dan Desain Sistem. Pembahasan akan meliputi implementasi sistem, konfigurasi *hyperparameter*, implementasi *Bidirectional Encoder Representations from Transformers*, serta aplikasi berupa website yang telah dibuat. Tabel 4.1 merupakan hubungan daftar segmen program dan desain sistem.

Tabel 4.1 Hubungan Segmen Program dan Desain Sistem

Segmen	Flowchart	Keterangan
Segmen Program 4.1	3.2	Melakukan pengolahan data yang akan digunakan dalam model untuk mempermudah pembacaan data ke dalam sistem
Segmen Program 4.2	3.3	Melakukan *preprocessing* sederhana terhadap dataset dengan mengubah tiap kata menjadi huruf kecil serta menghapus spasi, tab, dan *newline* yang berlebihan
Segmen Program 4.3	-	Menghubungkan *google colaboratory* dengan *google drive*
Segmen Program 4.4	-	Melakukan inisialisasi *hyperparameter* pada *argument parser extractive summarizer*
Segmen Program 4.5	-	Inisialisasi *main argument parser*
Segmen Program 4.6	-	Membuat *class* BERT data yang berguna untuk menampung *tokenizer*, model, dan *special token* dari BERT
Segmen Program 4.7	-	Fungsi *pad batch collate* pada *dataloder*
Segmen Program 4.8	3.4	Melakukan pembuatan *dataloader* untuk proses pengambilan data yang diperlukan untuk proses *training*, evaluasi, dan *testing*
Segmen Program 4.9	3.4, 3.5	Implementasi proses *training*, evaluasi, dan *testing*

Segmen Program 4.10	3.6	Proses implementasi susunan *transformer encoder* untuk melakukan klasifikasi pada layer *summarization*
Segmen Program 4.11	3.1	Evaluasi dengan menggunakan ROUGE score
Segmen Program 4.12	-	Melakukan prediksi ringkasan berita

4.1 Pengolahan dataset

Pada bagian ini dataset akan dilakukan pengolahan terlebih dahulu dengan mengambil bagian ‘paragraphs’, ‘gold_labels’, dan ‘summary’. Tujuan dari proses ini adalah untuk mempermudah dalam proses training ke dalam model yang dibuat. Tahapan ini akan dilakukan pada setiap data *train*, *dev*, dan *test*. Kemudian, pada bagian ini akan digunakan delimiter ‘<q>’ untuk pemisah antar kalimat pada bagian ‘paragraphs’ dan ‘summary’. Sama halnya dengan bagian ‘gold_labels’ yang dipakai sebagai pemisah antar label ringkasan.

Segmen Program 4.1 Proses pengolahan data

```
import os
import shutil
import json
from time import time
import torch
from datetime import timedelta
from argparse import ArgumentParser

parser = ArgumentParser(description='Data Preparation for Indonesian News
Summarization')
parser.add_argument('--original_data_dir',type=str,default='./indosum_original', help='path
to original dataset')
parser.add_argument('--save_dir', type=str, default='./prepared_data/indo_news_',
help='path to save prepared data')
args = parser.parse_args()

def read(fname):
   data = []
   for line in open(fname, 'r').readlines():
      datum = json.loads(line)
      label=''
      source=''
      target= ''

      # Source document and gold labels
      for idx in range(len(datum['paragraphs'])):
```

```
            for idy in range(len(datum['paragraphs'][idx])):
                for idz in range(len(datum['paragraphs'][idx][idy])):
                    source+=datum['paragraphs'][idx][idy][idz]+' '
                source+='<q>'
                label+=str(int(datum['gold_labels'][idx][idy])) + '<q>'

        # Gold summaries
        for idx in range(len(datum['summary'])):
            for idy in range(len(datum['summary'][idx])):
                target+=datum['summary'][idx][idy] + ' '
            target+='<q>'

        source = source[:-3]
        target = target[:-3]
        label = label[:-3]
        data.append((source, target, label))
    return data

def process(path, save_dir):
    dataset = []
    data = read(path)
    corpus_type = path.split('/')[-1].split('.')[0]
    fold = path.split('/')[-1].split('.')[1]
    start_time = time()
    for idx, datum in enumerate(data):
        source, target, sent_labels = datum
        source = preprocessing(source)
        target = '<q>'.join([' '.join(sent.split()) for sent in target.split('<q>')])  # Untuk
menghilangkan spasi yang berlebihan
        b_data_dict = {"source": source,"labels": sent_labels, "target": target}
        dataset.append(b_data_dict)
        if (idx+1) % 500 == 0:
            end_time = time()
            print(f'{(idx+1)} data processed | {corpus_type}-{fold} | runtime: {end_time-
start_time} seconds')
            start_time = end_time

    if len(dataset) > 0:
        pt_file = save_dir + "{:s}.indonews.bert.pt".format(corpus_type)
        torch.save(dataset, pt_file)

# To speed up the processing data, data will be saved as .pt format
fold = [1,2,3,4,5]
start = time()
for i in fold:
    save_dir = args.save_dir + str(i) + '/' #path to processed data
    print('Create ', save_dir)
    if os.path.exists(save_dir): # check if path exist
        shutil.rmtree(save_dir)  # remove file inside directory recursively
```

```
    os.makedirs(save_dir) # create directory
    process(args.original_data_dir + '/train.0'+str(i)+'.jsonl', save_dir)
    process(args.original_data_dir + '/dev.0'+str(i)+'.jsonl', save_dir)
    process(args.original_data_dir + '/test.0'+str(i)+'.jsonl', save_dir)
end = time()
elapsed_time = timedelta(seconds=end-start)
print(f'Elapsed time: {elapsed_time}')
```

4.2 Preprocessing

Preprocessing yang akan diterapkan pada dataset adalah mengubah tiap kata dalam kalimat menjadi huruf kecil, menghilangkan karakter spasi, tab, dan *newline* yang berlebihan. Pada *preprocessing* ini, delimiter untuk pemisah antar kalimat akan menggunakan '<q>'.

Segmen Program 4.2 Fungsi *preprocessing* data

```
def preprocessing(text):
    # Delimiter of split sentence
    delimiter = '<q>'

    # Case folding (lowercase text)
    text = text.lower()

    # Clean excessive space, tab, and newline
    text = delimiter.join([' '.join(sent.split()) for sent in text.split(delimiter)])

    return text
```

4.3 Konfigurasi google colaboratory

Proses awal yang dilakukan sebelum melakukan implementasi proses training dari model BERT adalah menghubungkan *google colaboratory* dengan *google drive*. Tujuan dari proses ini adalah untuk mempermudah dalam pembacaan data sehingga tidak perlu melakukan inisialisasi data berulang kali pada *google colaboratory*. Proses penghubungan *google colaboratory* dengan *google drive* dapat dilihat pada Segmen Program 4.3.

Segmen Program 4.3 Menghubungkan *google colaboratory* dengan *google drive*

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

Setelah menghubungkan *google colaboratory* dengan *google drive*, dilanjutkan dengan menghubungkan *google colaboratory* dengan GPU. Dalam menghubungkan langkah yang dilakukan adalah dengan cara memilih menu *Runtime* > *Change runtime type* > *Hardware*

accelerator > ubah *hardware accelerator* menjadi GPU. Mengenai keseluruhan *setup* sebelum memulai training pada *google colaboratory* dapat dilakukan dengan *command line*:

Keterangan setup	Command line
Cek GPU yang dipinjamkan *google colaboratory*	!nvidia-smi
Install library pytorch-lightning versi 1.4.9	!pip install pytorch_lightning==1.4.9
Install library transformers	!pip install transformers
Install library untuk *logger wandb*	!pip install wandb
Install dan *setup library pyrouge*	!git clone https://github.com/andersjo/pyrouge.git /content/rouge !git clone https://github.com/bheinzerling/pyrouge /content/pyrouge %cd /content/pyrouge !python setup.py install !pyrouge_set_rouge_path '/content/rouge/tools/ROUGE-1.5.5' %cd /content/rouge/tools/ROUGE-1.5.5/data !rm "WordNet-2.0.exc.db" !perl ./WordNet-2.0-Exceptions/buildExeptionDB.pl ./WordNet-2.0-Exceptions ./smart_common_words.txt ./WordNet-2.0.exc.db !cpan install XML::DOM !python -m pyrouge.test

4.4 Inisialisasi hyperparameter

Sebelum melakukan training, perlu dilakukan inisialisasi terhadap *hyperparameter* yang akan digunakan oleh model. Proses untuk inisialisasi *hyperparameter* beserta *argument* lain dari model *extractive summarizer* akan dilakukan dengan menggunakan *argument parser* seperti yang dapat dilihat pada Segmen Program 4.4. Kemudian, untuk *argument parser* ini akan dipanggil oleh *parent parser* pada bagian *main*. *Argument parser* yang terdapat pada bagian *main* ini akan digunakan sebagai input *argument* dari *trainer pytorch lightning*. Mengenai

argument parser pada bagian *main* dapat dilihat pada Segmen Program 4.5. Mengenai penjelasan untuk tiap *argument* dapat dilihat pada bagian *help* dari tiap *argument*.

Segmen Program 4.4 Inisialisasi *hyperparameter* pada *argument parser extractive summarizer*

```
  @staticmethod
  def add_model_specific_args(parent_parser):
    """Arguments specific to this model"""
    parser = ArgumentParser(parents=[parent_parser],description='Extractive
Summarization on Indonesian News using BERT')
    parser.add_argument("--save_dir", type=str, default="./bert_checkpoints",
help="Directory path to save model")
    parser.add_argument("--ref_summary", type=int, default=0, help="Reference summary
for scoring evaluation (0 = gold label, 1 = gold summary)")
    parser.add_argument(
      "--model_name_or_path",
      type=str,
      default="bert-base-multilingual-uncased",
      help="Path to pre-trained model or shortcut name.",
    )
    parser.add_argument(
      "--model_type",
      type=str,
      default="bert",
      help="Used model type.",
    )
    parser.add_argument(
      "--max_seq_length",
      type=int,
      default=512,
      help="The maximum sequence length of BERT and transformer model.",
    )
    parser.add_argument(
      "--data_path", type=str, default='./prepared_data_basic/indo_news_',
help="Directory containing used data."
    )
    parser.add_argument(
      "--pooling_mode",
      type=str,
      default="sent_rep_tokens",
      help="Convert word vectors to sentence embeddings.",
    )
    parser.add_argument(
      "--num_frozen_steps",
      type=int,
      default=0,
      help="Freeze (don't train) the word embedding model for this many steps.",
    )
    parser.add_argument(
```

```
        "--batch_size",
        default=8,
        type=int,
        help="Batch size per GPU/CPU for training/evaluation/testing.",
    )
    parser.add_argument(
        "--dataloader_num_workers",
        default=2,
        type=int,
        help="""The number of workers to use when loading data. A general place to
        start is to set num_workers equal to the number of CPU cores used machine.""",
    )
    parser.add_argument(
        "--no_use_token_type_ids",
        action="store_true",
        help="Set to not train with `token_type_ids`.",
    )
    parser.add_argument(
        "--classifier",
        type=str,
        default="transformer_position",
        help="""Which classifier/encoder to use to reduce the hidden dimension of the
sentence vectors""",
    )
    parser.add_argument(
        "--classifier_dropout",
        type=float,
        default=0.1,
        help="The value for the dropout layers in the classifier.",
    )
    parser.add_argument(
        "--classifier_transformer_num_layers",
        type=int,
        default=2,
        help='The number of layers for the `transformer` classifier.',
    )
    parser.add_argument(
        "--train_name",
        type=str,
        default="train",
        help="name for set of training files on disk.",
    )
    parser.add_argument(
        "--val_name",
        type=str,
        default="dev",
        help="name for set of validation files on disk.",
    )
    parser.add_argument(
```

```
        "--test_name",
        type=str,
        default="test",
        help="name for set of testing files on disk.",
    )
    parser.add_argument(
        "--test_k",
        type=int,
        default=3,
        help="The `k` parameter to chose top k predictions from the model for evaluation
scoring (default: 3)",
    )
    parser.add_argument(
        "--n_gram_blocking",
        type=int,
        default=3,
        help="number of n-gram blocking for testing"
    )
    parser.add_argument(
        "--no_test_block_ngrams",
        action="store_true",
        help="Disable n-gram blocking when calculating ROUGE scores during testing.",
    )

    return parser
```

Segmen Program 4.5 Inisialisasi *main argument parser*

```
import logging
from pytorch_lightning import Trainer
from extractive import ExtractiveSummarizer
from argparse import ArgumentParser
from pytorch_lightning.loggers import WandbLogger
from pytorch_lightning.callbacks import LearningRateMonitor
from pytorch_lightning import seed_everything


logger = logging.getLogger(__name__)

def main(args):
  if args.seed:
    seed_everything(args.seed, True)
    args.deterministic = True

  model = ExtractiveSummarizer(hparams=args)
  lr_logger = LearningRateMonitor()

  type_used_data = args.data_path.split('/')[-2].split('_')[-1]
  if args.no_use_token_type_ids:
```

```
        temp_token_type_ids = "no-token-type-ids"
    else:
        temp_token_type_ids = "use-token-type-ids"
    temp_pooling_mode = args.pooling_mode.replace('_','-')

    if args.use_logger == "wandb":
        wandb_name =
f"{args.model_name_or_path}_{type_used_data}_{temp_token_type_ids}_{temp_pooling_
mode}_{args.classifier}_{args.classifier_transformer_num_layers}_{args.seed}_{args.learning
_rate}_{args.classifier_dropout}_{args.batch_size}_{args.max_epochs}"
        wandb_logger = WandbLogger(
            name=wandb_name,project=args.wandb_project, log_model=(not
args.no_wandb_logger_log_model)
        )
        args.logger = wandb_logger

    args.callbacks = [lr_logger]
    trainer = Trainer.from_argparse_args(args)

    if args.do_train:
        trainer.fit(model)
    if args.do_test:
        trainer.test(model)


if __name__ == "__main__":
    parser = ArgumentParser(add_help=False)
    parser.add_argument(
        "--default_root_dir", type=str, default=None, help="Default path for logs and weights.",
    )
    parser.add_argument(
        "--weights_save_path",
        type=str,
        default=None,
        help="""Where to save weights if specified. Will override `--default_root_dir` for
        checkpoints only.""",
    )
    parser.add_argument(
        "--learning_rate",
        default=1e-5,
        type=float,
        help="The initial learning rate for the optimizer.",
    )
    parser.add_argument(
        "--min_epochs",
        default=1,
        type=int,
        help="Limits training to a minimum number of epochs",
    )
```

```
    parser.add_argument(
        "--max_epochs",
        default=4,
        type=int,
        help="Limits training to a max number number of epochs",
    )
    parser.add_argument(
        "--min_steps",
        default=None,
        type=int,
        help="Limits training to a minimum number number of steps",
    )
    parser.add_argument(
        "--max_steps",
        default=None,
        type=int,
        help="Limits training to a max number number of steps",
    )
    parser.add_argument(
        "--accumulate_grad_batches",
        default=1,
        type=int,
        help="""Accumulates grads every k batches.""",
    )
    parser.add_argument(
        "--check_val_every_n_epoch",
        default=1,
        type=int,
        help="Check val every n train epochs.",
    )
    parser.add_argument(
        "--gpus",
        default=1,
        type=int,
        help="Number of GPUs to train on or Which GPUs to train on. (-1 = all gpus, 1 = only
using one GPU)",
    )
    parser.add_argument(
        "--gradient_clip_val", default=1.0, type=float, help="Gradient clipping value (default
gradient clipping algorithm is set to 'norm' and clip global norm to <=1.0)"
    )
    parser.add_argument(
        "--fast_dev_run",
        action="store_true",
        help="Runs 1 batch of train, test and val to find any bugs (ie: a sort of unit test).",
    )
    parser.add_argument(
        "--limit_train_batches",
        default=1.0,
```

```
        type=float,
        help="How much of training dataset to check. Useful when debugging or testing
something that happens at the end of an epoch.",
    )
    parser.add_argument(
        "--limit_val_batches",
        default=1.0,
        type=float,
        help="How much of validation dataset to check. Useful when debugging or testing
something that happens at the end of an epoch.",
    )
    parser.add_argument(
        "--limit_test_batches",
        default=1.0,
        type=float,
        help="How much of test dataset to check.",
    )
    parser.add_argument(
        "--precision",
        type=int,
        default=32,
        help="Full precision (32). Can be used on CPU, GPU or TPUs.",
    )
    parser.add_argument(
        "--seed",
        type=int,
        default=1,
        help="Seed for reproducible results and fold id.",
    )
    parser.add_argument(
        "--profiler",
        default="simple",
        type=str,
        help="To profile individual steps during training and assist in identifying bottlenecks.",
    )
    parser.add_argument(
        "--progress_bar_refresh_rate",
        default=50,
        type=int,
        help="How often to refresh progress bar (in steps).",
    )
    parser.add_argument(
        "--num_sanity_val_steps",
        default=0,
        type=int,
        help="Sanity check runs n batches of val before starting the training routine. This
catches any bugs in your validation without having to wait for the first validation check.",
    )
    parser.add_argument(
```

```
      "--val_check_interval",
      default=1.0,
      help="How often within one training epoch to check the validation set. Can specify as
float or int. Use float to check within a training epoch. Use int to check every n steps
(batches).",
   )
   parser.add_argument(
      "--use_logger",
      default="wandb",
      type=str,
      help="Which program to use for logging. Default to `wandb`.",
   )
   parser.add_argument(
      "--wandb_project",
      default="skripsi",
      type=str,
      help="The wandb project to save training runs.",
   )
   parser.add_argument(
      "--do_train", action="store_true", help="Run the training procedure."
   )
   parser.add_argument(
      "--do_test", action="store_true", help="Run the testing procedure."
   )
   parser.add_argument(
      "--load_checkpoint",
      default=None,
      type=str,
      help="Loads the model weights and hyperparameters from a given checkpoint.",
   )
   parser.add_argument(
      "--no_wandb_logger_log_model",
      action="store_true",
      help="Only applies when using the `wandb` logger. Set this argument to NOT save
checkpoints in wandb directory to upload to W&B servers.",
   )
   parser.add_argument(
      "--adam_epsilon", default=1e-8, type=float, help="Epsilon for Adam optimizer.",
   )
   parser.add_argument("--weight_decay", default=1e-2, type=float, help="weight decay for
adam")
   parser.add_argument(
      "--optimizer_type",
      default="adamw",
      type=str,
      help="""Which optimizer to use: `adamw` (default)""",
   )
   parser.add_argument(
      "--warmup_steps",
```

```
        default=0,
        type=int,
        help="Linear warmup over warmup_steps.",
    )
    parser.add_argument(
        "--use_scheduler",
        default="linear",
        type=str,
        help="`linear`: Use a linear schedule that inceases linearly over `--warmup_steps` to `--
learning_rate` then decreases linearly for the rest of the training process.",
    )
    parser.add_argument(
        "--log",
        dest="logLevel", # name of the attribute to be added to the object
        default="INFO",
        choices=["DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"],
        help="Set the logging level (default: 'Info').",
    )

    parser = ExtractiveSummarizer.add_model_specific_args(parser)
    main_args = parser.parse_args()

    # Setup logging config
    logging.basicConfig(
        format="%(asctime)s|%(name)s|%(levelname)s> %(message)s",
        level=logging.getLevelName(main_args.logLevel),
    )

    # Train and Test
    main(main_args)
```

4.5 Bert Data dan Dataloader

Setelah melakukan inisialisasi untuk *hyperparameter*, akan dilakukan pembuatan *class* BertData yang akan menyimpan atribut dan fungsi-fungsi yang akan digunakan pada model BERT seperti *tokenizer*, model, token spesial, dan token id yang digunakan. Proses pembuatan *class* BertData dapat dilihat pada Segmen Program 4.6. Kemudian, BertData yang telah dibuat akan digunakan untuk mendapatkan *input feature* yang diperlukan untuk melatih model. *Input feature* yang akan digunakan ke dalam model BERT adalah setelah dilakukan pembuatan *dataloader*. Untuk menghasilkan *input feature* pada saat proses pembuatan *dataloader* akan digunakan fungsi *pad batch collate* karena input data yang dimasukkan berupa *list dictionary*. Fungsi *pad batch collate* dapat dilihat pada Segmen Program 4.7. Kemudian, mengenai proses pembuatan *dataloader* untuk training, validasi, dan testing dapat dilihat pada Segmen Program 4.8.

Segmen Program 4.6 BertData

```
class BertData():
   def __init__(self, pre_trained_bert_model):
      self.model = BertModel.from_pretrained(pre_trained_bert_model)
      self.tokenizer = BertTokenizer.from_pretrained(pre_trained_bert_model)
      self.sep_token = '[SEP]'
      self.cls_token = '[CLS]'
      self.pad_token = '[PAD]'
      self.sep_vid = self.tokenizer.vocab[self.sep_token]
      self.cls_vid = self.tokenizer.vocab[self.cls_token]
      self.pad_vid = self.tokenizer.vocab[self.pad_token]

   def get_input_features(self, source, target, labels, min_seq_length=1,
max_seq_length=512, delimiter='<q>'):
      source = source.split(delimiter)
      original_src_txt = source

      source = [sent.strip() for sent in source]
      idxs = [i for i, s in enumerate(source) if (len(s) >= min_seq_length)]

      labels = labels.split('<q>')
      labels = [int(l) for l in labels]

      tokenized_source = self.tokenizer.tokenize(self.sep_token.join(source))
      temp = []
      tokens = []
      flag = True
      for sub_token in tokenized_source:
         if flag:
            tokens.append(self.cls_token)
            flag = False

         tokens.append(sub_token)

         if sub_token == self.sep_token:
            temp.append(tokens)
            tokens = []
            flag = True

      # Check exceeded length (max. token length = 512)
      res = []
      total_len = 0
      for idx, l in enumerate(temp):
         total_len+=len(l)
         if total_len > max_seq_length:
            break
         res.append(temp[idx])
```

```
        source_subtokens = [t for s in res for t in s]
        input_ids = self.tokenizer.convert_tokens_to_ids(source_subtokens)

        # Segments ids / Token type ids
        _segs = [-1] + [i for i, t in enumerate(input_ids) if t == self.sep_vid]
        segs = [_segs[i] - _segs[i - 1] for i in range(1, len(_segs))]
        segments_ids = []
        for i, s in enumerate(segs):
            if (i % 2 == 0):
                segments_ids += s * [0]
            else:
                segments_ids += s * [1]

        cls_ids = [i for i, t in enumerate(input_ids) if t == self.cls_vid]
        labels = labels[:len(cls_ids)]
        source = [original_src_txt[i] for i in idxs]

        return input_ids, segments_ids, labels, cls_ids, source, target

class DatasetIndoNews(torch.utils.data.Dataset):
    def __init__(self, doc, bert, max_seq_length=512):
        super(DatasetIndoNews,self).__init__()
        self.doc = doc
        self.bert = bert
        self.features = self.get_input_features(doc)

    def get_input_features(self, doc):
        input_features = []
        for item in doc:
            input_ids, token_type_ids, labels, cls_ids, source, target =
self.bert.get_input_features(item['source'], item['target'], item['labels'])
            bert_features = {"input_ids": input_ids, "token_type_ids": token_type_ids,
                    "labels": labels, "sent_rep_token_ids": cls_ids,
                    "source":source, "target":target}
            input_features.append(bert_features)
        return input_features

    def get_bert(self):
        return self.bert

    def get_doc(self, idx):
        return self.doc[idx]

    def get_len_doc(self):
        return len(self.doc)

    def __getitem__(self, idx):
        return self.features[idx]
```

```
    def __len__(self):
        return len(self.features)
```

Segmen Program 4.7 Fungsi *pad batch collate*

```
def pad_batch_collate(batch):
    elem = batch[0]
    final_dictionary = {}

    # Iterate through all key dictionary
    for key in elem:

        # For each data key in batch append to list of feature
        feature_list = [d[key] for d in batch]
        if key == "sent_rep_token_ids":

            feature_list = pad(feature_list, -1)
            sent_rep_token_ids = torch.tensor(feature_list, dtype=torch.long)

            sent_rep_mask = ~(sent_rep_token_ids == -1)
            sent_rep_token_ids[sent_rep_token_ids == -1] = 0

            final_dictionary["sent_rep_token_ids"] = sent_rep_token_ids
            final_dictionary["sent_rep_mask"] = sent_rep_mask
            continue
        if key == "input_ids":
            input_ids = feature_list

            # Attention
            # The mask has 1 for real tokens and 0 for padding tokens. Only real
            # tokens are attended to.
            attention_mask = [[1] * len(ids) for ids in input_ids]

            input_ids_width = max([len(ids) for ids in input_ids])
            input_ids = pad(input_ids, 0, width=input_ids_width)
            input_ids = torch.tensor(input_ids, dtype=torch.long)

            attention_mask = pad(attention_mask, 0)
            attention_mask = torch.tensor(attention_mask, dtype=torch.long)

            final_dictionary["input_ids"] = input_ids
            final_dictionary["attention_mask"] = attention_mask

            continue

        if key in ("source", "target"):
            final_dictionary[key] = feature_list
            continue
```

```
        if key in ("labels", "token_type_ids"):
            feature_list = pad(feature_list, 0)

        feature_list = torch.tensor(feature_list, dtype=torch.long)
        final_dictionary[key] = feature_list

    return final_dictionary
```

Segmen Program 4.8 Proses pembuatan *dataloader*

```
    def train_dataloader(self):
        if self.train_dataloader_object:
            return self.train_dataloader_object
        if not hasattr(self,"datasets"):
            self.prepare_data()
        self.global_step_tracker = 0

        train_dataset = self.datasets[self.hparams.train_name]
        train_dataloader = DataLoader(
            train_dataset,
            num_workers=self.hparams.dataloader_num_workers,
            batch_size=self.hparams.batch_size,
            collate_fn=self.pad_batch_collate,
            shuffle=True
        )
        self.train_dataloader_object = train_dataloader
        return train_dataloader

    def val_dataloader(self):
        valid_dataset = self.datasets[self.hparams.val_name]
        valid_dataloader = DataLoader(
            valid_dataset,
            num_workers=self.hparams.dataloader_num_workers,
            batch_size=self.hparams.batch_size,
            collate_fn=self.pad_batch_collate,
            shuffle=False
        )
        return valid_dataloader

    def test_dataloader(self):
        test_dataset = self.datasets[self.hparams.test_name]
        test_dataloader = DataLoader(
            test_dataset,
            num_workers=self.hparams.dataloader_num_workers,
            batch_size=self.hparams.batch_size,
            collate_fn=self.pad_batch_collate,
            shuffle=False
        )
        return test_dataloader
```

Mengenai salah satu contoh *input feature* yang akan diteruskan ke dalam model BERT dapat dilihat pada Segmen Data 4.1. Lalu, untuk keterangan lebih lanjut tentang *input feature* BERT *summarization* yang dihasilkan pada *dataloader* dapat dilihat pada Tabel 4.2.

Segmen Data 4.1 Contoh *input features* BERT untuk proses training

```
{
'input_ids': [3, 22087,  4403,    17,  4403,  9282, 19433,  1836,  2699,  3761,
        43,  1485,  6887,    18,    4,    3, 19433, 22087, 20872,  3761,
        43,    28,    18,    20,    12,  5446,    13,  1709,  3956, 26997,
     22958,  1889,  2115, 12422,  1620,  4766,    18,    4,    3,  1684,
      1841,  4440,  2390,  1485,  3956,  7300,  1540,  4433,  1545,  2699,
      2442,    16,  2936,  1560,  6368, 14664,  1501, 17002,  7756,    18,
         4,    3, 22087, 20872,  3761,    43,    28,    18,    20,    12,
      5446,    13,  5346,  6672,    65,   962,  1534,    12, 14588,   946,
        66,  8040, 13673,  4758,    13,    28, 16870,  1501,  6099,  1617,
     18098, 11795,  9091,  1502,  8596, 10222,  6113,   959, 25799,  1497,
      2528,  3695,  5750, 14214,  4831,  1485,    21,    18,    24,  7582,
       951,    18,    4,    3,  1624,  1560,  6287,  1519,    16,  1684,
      2778,  6944,    22, 17964,  1501, 10529,  7163,  2635,  2542, 17964,
      1497,  1708,  1485,  1562, 21332,  1545,  5190,  8689,   933,  1967,
     25804, 17964,    18,    4,    3, 19433,  1497,  1798,  1777, 21910,
      1485,  6887,  1540,  3375,  1545,  5052, 18279,    27,    18,    21,
     19685,  1534,   930,    16,  1501,  3669,  1713,  5190,  3728, 23278,
        18,    4,    3,  4433,  1545,  3956,  4766,    16, 20872,  3761,
        43,    28,    18,    20,    12,  5446,    13,  2101,  4440,  1485,
      1823,  7756,    16,  1485,  2420,  2338,  1677,  6668,  7756,  2408,
        28,  3349,  1545,  1675, 10486,  7885,    48,    19,    21,    18,
        29,    16,  1501,  6869,  7756,  2616,    25,  3349,    18,    4,
         3,  1624,  1797,  2873,    16,  3956, 20872,  3761, 12422,  2461,
      4112,  1545,  7756,  3615,    25,  3349,  2327, 15378, 15900,  1501,
      7756,  2616,  2616,  1821,    22,  3349,    18,    4,    3,  4153,
      1704,  1533,  7056,  1559, 20872,  3761,    43,    28,    18,    20,
        12,  5446,    13,  5408,    24,    49, 12538,   929,    16,  7371,
        17,  3801,    43,    19,    44,    19,    49,    19,    56,    16,
     27079,    24,    18,    22,    16,  7360, 12609,    23,    18,    25,
      6546,    16,  1501, 22949,    15, 31318,  4294,   944,    18,    4,
         3, 19433,  7542, 18100,    18,    21,  1060, 10002,   947,    18,
        21,  1060,   963,    18,    29,  6546,  1501,  3314,  5305,   947,
      7585,  1487,  1540,  2891,  1716,  8666,  1501,  3314,  8036,  3956,
      1620,  4766,    16,  3464,  1686, 14664,  1497,  4474,  1614, 16740,
     21395,  2071,  1497, 17834,  1614,  1716,  1819,    18,    4,    3,
      2530,  3524, 22087,    18,    4,    3, 11985,  5470, 25852,    18,
      2593,  1581, 17440,  3524, 14249,  1562,  1501,  9602,  3279,    18,
         4,    0,    0,    0,    0,    0,    0,    0,    0,    0,
```

```
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0],
'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'labels': [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'sent_rep_token_ids': [0, 15, 38, 61, 113, 144, 172, 220, 248, 300, 349, 355,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
'sent_rep_mask': [True, True, True, True, True, True, True, True, True, True, True, True,
False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False]
}
```

Tabel 4.2 *Input features* model BERT untuk proses training

#	Feature	Keterangan
1	input_ids	id token dari model BERT yang telah diinisialisasi sebelumnya oleh pembuat model. Pada input_ids ini juga akan berisi id dari *special token* [CLS], [SEP], dan [PAD]
2	attention_mask	Berisi *integer* berupa 0 dan 1 untuk membedakan token *padding* dan token sebenarnya dari id token. 0 merupakan token *padding* sedangkan 1 merupakan token sebenarnya (selain token *padding*)
3	token_type_ids	Berisi *integer* berupa 0 dan 1 untuk membedakan antara kalimat posisi indeks ganjil dan genap.
4	labels	Berupa list yang berisi integer berupa 0 dan 1 sebagai label untuk referensi ekstraktif
5	sent_rep_token_ids	Berisi *integer* berupa indeks token untuk *special token* [CLS]
6	sent_rep_mask	Berisi *boolean* berupa True dan False untuk membedakan token *padding* dan token sebenarnya dari indeks *special token* [CLS]. False menunjukkan token *padding* sedangkan True menunjukkan token sebenarnya (selain token *padding*)

4.6 Training dan Testing

Proses training dimulai dengan menginisialisasikan class *trainer* dari *pytorch lightning* dengan input *main argument parser* dari Segmen Program 4.5. *Pytorch lightning* akan menampung model, *optimizer*, dan *train/val/test step*. Kemudian, *pytorch lightning* akan

memulai proses training dengan mendefinisikan *action* yang diperlukan untuk proses training. List *action* yang digunakan pada modul *pytorch lightning* terdiri atas:

Action	Keterangan
setup	Untuk melakukan pengaturan sebelum proses *training* dan *testing*
prepare_data	Menyiapkan data yang akan digunakan sebelum diteruskan ke dalam *dataloader*
train_dataloader	Membuat *dataloader* untuk pelatihan model
val_dataloader	Membuat *dataloader* untuk melakukan evaluasi dari proses *training*
test_dataloader	Membuat *dataloader* untuk melakukan *testing* dari model yang sudah dilatih
configure_optimizers	Melakukan konfigurasi untuk *optimizer* dan *scheduler* yang dipakai untuk melatih model
training_step	Melakukan proses *training* menggunakan *optimizer* dan *scheduler* yang sudah dikonfigurasikan pada configure_optimizers
training_epoch_end	Akhir tiap *epoch training* untuk menerima output dari training_step
validation_step	Melakukan proses validasi/evaluasi dari training
validation_epoch_end	Akhir tiap *epoch* validasi/evaluasi untuk menerima output dari validation_step
compute_loss	Melakukan perhitungan *loss* antara output model dengan label target
test_step	Melakukan proses *testing* dari model yang sudah dilatih
test_epoch_end	Akhir tiap *epoch* test untuk menerima output dari test_step
foward	Melakukan *forward* dari model dengan meneruskan input ke dalam model untuk dicari tahu nilai dari outputnya

Selain itu, *command* yang perlu dilakukan berulang kali atau dikenal dengan istilah *boilerplate* akan ditangani secara otomatis oleh *pytorch lightning* untuk membantu mempermudah dalam melakukan penelitian sehingga peneliti dapat fokus terhadap bagian utama dari proses *training* dan *testing*. Beberapa *command* tersebut diantaranya:

Command code	Keterangan
.to(device)	Meletakkan *batch* dan komputasi ke dalam *device* yang digunakan
.set_grad_enabled()	Mengaktifkan gradien untuk proses training
.train()	Mengubah ke mode training
.eval()	Mengubah ke mode eval
.zero_grad()	Mengatur gradien untuk semua parameter model menjadi nol
.backward()	Melakukan *backward* untuk menghitung gradien dalam model
.step()	Melakukan *update parameter* dari model

Pada setiap akhir *epoch training* akan dilakukan validasi atau evaluasi yang bertujuan untuk mengetahui seberapa baik model belajar. Mengenai segmen program yang digunakan untuk melakukan keseluruhan proses *training*, evaluasi, dan *testing* pada model yang sudah dibuat dapat dilihat pada Segmen Program 4.9.

Segmen Program 4.9 Implementasi proses *training*, *validasi*, dan *testing*

```
def forward(
      self,
      input_ids,
      attention_mask,
      sent_rep_mask=None,
      token_type_ids=None,
      sent_rep_token_ids=None,
      **kwargs,
   ):
      inputs = {
         "input_ids": input_ids,
         "attention_mask": attention_mask,
      }
```

```
    if not self.hparams.no_use_token_type_ids:
        inputs["token_type_ids"] = token_type_ids

    outputs = self.word_embedding_model(**inputs, **kwargs)
    word_vectors = outputs[0]

    sents_vec, mask = self.pooling_model(
        word_vectors=word_vectors,
        sent_rep_token_ids=sent_rep_token_ids,
        sent_rep_mask=sent_rep_mask,
    )

    sent_scores = self.encoder(sents_vec, mask)
    return sent_scores, mask

def unfreeze_word_embedding_model(self):
    for param in self.word_embedding_model.parameters():
        param.requires_grad = True

def freeze_word_embedding_model(self):
    for param in self.word_embedding_model.parameters():
        param.requires_grad = False

def compute_loss(self, outputs, labels, mask):
    loss = self.loss_func(outputs, labels.float())

    # Set all padding values to zero
    loss = loss * mask.float()

    # Add up all the loss values for each sequence (including padding because
    # padding values are zero and thus will have no effect)
    sum_loss_per_sequence = loss.sum(dim=1)

    # Count the number of losses that are not padding per sequence
    num_not_padded_per_sequence = mask.sum(dim=1).float()

    # Find the average loss per sequence
    average_per_sequence = sum_loss_per_sequence / num_not_padded_per_sequence

    # Get the sum of the average loss per sequence
    sum_avg_seq_loss = average_per_sequence.sum()

    # Get the mean of `average_per_sequence`
    batch_size = average_per_sequence.size(0)
    mean_avg_seq_loss = sum_avg_seq_loss / batch_size

    return mean_avg_seq_loss
```

```
    def setup(self, stage):
        if stage == "fit":
            self.word_embedding_model = self.bert.get_model_with_config(
                self.hparams.model_name_or_path,
model_config=self.word_embedding_model.config
            )
            if self.checkpoint is not None:
                self.load_state_dict(self.checkpoint['model_state_dict'])
                self.epoch = self.checkpoint['epoch'] + 1
                logger.info("Epoch start from %s", self.epoch)

                self.list_train_loss_epoch = self.checkpoint['train_histories']['loss']
                self.list_val_loss_epoch = self.checkpoint['val_histories']['loss']
                self.min_loss = min(self.list_val_loss_epoch)
            else:
                logger.info("Training without checkpoint")

        if stage == "test":
            if self.checkpoint is None:
                logger.info("Need to specify path checkpoint model to test model")
                sys.exit(1)
            self.load_state_dict(self.checkpoint['model_state_dict'])


    def prepare_data(self):
        datasets = {}
        data_splits = [
            self.hparams.train_name,
            self.hparams.val_name,
            self.hparams.test_name,
        ]

        for corpus_type in data_splits:
            full_path = self.hparams.data_path + str(self.hparams.seed) + "/" + corpus_type +
".indonews.bert.pt"
            torch_data = torch.load(full_path)
            data = [x for x in torch_data]
            max_seq_length = min(round(get_average_length(self.bert, data)),
self.hparams.max_seq_length)
            datasets[corpus_type] = DatasetIndoNews(data, self.bert, max_seq_length)

        self.datasets = datasets
        self.pad_batch_collate = pad_batch_collate


    def train_dataloader(self):
        if self.train_dataloader_object:
            return self.train_dataloader_object
        if not hasattr(self,"datasets"):
```

```
        self.prepare_data()
    self.global_step_tracker = 0

    train_dataset = self.datasets[self.hparams.train_name]
    train_dataloader = DataLoader(
        train_dataset,
        num_workers=self.hparams.dataloader_num_workers,
        batch_size=self.hparams.batch_size,
        collate_fn=self.pad_batch_collate,
        shuffle=True
    )
    self.train_dataloader_object = train_dataloader
    return train_dataloader

  def val_dataloader(self):
    valid_dataset = self.datasets[self.hparams.val_name]
    valid_dataloader = DataLoader(
        valid_dataset,
        num_workers=self.hparams.dataloader_num_workers,
        batch_size=self.hparams.batch_size,
        collate_fn=self.pad_batch_collate,
        shuffle=False
    )
    return valid_dataloader

  def test_dataloader(self):
    test_dataset = self.datasets[self.hparams.test_name]
    test_dataloader = DataLoader(
        test_dataset,
        num_workers=self.hparams.dataloader_num_workers,
        batch_size=self.hparams.batch_size,
        collate_fn=self.pad_batch_collate,
        shuffle=False
    )
    return test_dataloader

  def configure_optimizers(self):
    self.train_dataloader_object = self.train_dataloader()
    optimizer = AdamW(self.parameters(), lr=self.hparams.learning_rate,
                eps=self.hparams.adam_epsilon, weight_decay=self.hparams.weight_decay)

    last_epoch = -1

    # Check load checkpoint model
    if self.checkpoint:
        logger.info("Currently using loaded optimizer")
        optimizer.load_state_dict(self.checkpoint['optimizer_state_dict'])
        last_epoch = self.checkpoint['epoch'] - 1
```

```
        total_steps = len(self.train_dataloader_object) * self.hparams.max_epochs
        scheduler = {
            'scheduler': get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=self.hparams.warmup_steps, num_training_steps=total_steps,
last_epoch=last_epoch),
            'interval':'step'
            }

        return [optimizer], [scheduler]


    def training_step(self, batch, batch_idx):
        # Get batch information
        labels = batch["labels"]
        sources = batch["source"]

        # Delete labels, source, target so now batch contains everything to be inputted into the
model
        del batch["labels"]
        del batch['source']
        del batch['target']


        # If global_step has increased by 1:
        # Begin training the `word_embedding_model` after `num_frozen_steps` steps
        if (self.global_step_tracker + 1) == self.trainer.global_step:
            self.global_step_tracker = self.trainer.global_step

            if self.emd_model_frozen and (self.trainer.global_step >
self.hparams.num_frozen_steps):
                self.emd_model_frozen = False
                self.unfreeze_word_embedding_model()

        # Compute model forward (forward pass to compute output with mask by passing
batch data to the model)
        outputs, mask = self.forward(**batch)

        # Compute loss
        train_loss = self.compute_loss(outputs, labels, mask)
        outputs = torch.sigmoid(outputs)

        # For compute F1 ROUGE score
        system_summaries = []
        ref_summaries = []
        for idx, label in enumerate(labels):
            temp = ""
            for idy, l in enumerate(label):
                if l:
                    temp += sources[idx][idy]
```

```
                temp += '<q>'
            temp = temp[:-3]
            ref_summaries.append(temp)

        source_ids = (
            torch.argsort(outputs, dim=1, descending=True)
        )

        for idx, (source, source_ids, target) in enumerate(
            zip(sources, source_ids, ref_summaries)
        ):
            pos = []
            for sent_idx, i in enumerate(source_ids):
                if i >= len(source):
                    continue

                pos.append(i.item())

                if len(pos) == self.hparams.test_k:
                    break
            pos.sort()
            selected_sentences = "<q>".join([source[i] for i in pos])
            system_summaries.append(selected_sentences)


        if self.hparams.no_use_token_type_ids:
            temp_token_type_ids = "no-token-type-ids"
        else:
            temp_token_type_ids = "use-token-type-ids"
        model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
        self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
            model_name_or_path, temp_token_type_ids,
            self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
            self.hparams.classifier_dropout, self.hparams.batch_size, self.hparams.max_epochs
            )

        self.temp_train_gold = self.save_file + "/train_gold_" + str(self.epoch) + ".txt"
        self.temp_train_pred = self.save_file + "/train_pred_" + str(self.epoch) + ".txt"

        os.makedirs(self.save_file, exist_ok=True)

        for pred in system_summaries:
            self.all_pred_train += str(pred).strip() + "\n"
        for gold in ref_summaries:
            self.all_gold_train += str(gold).strip() + "\n"

        train_dict = {
            'epoch': self.epoch,
```

```
            'loss': train_loss,
            }
        for name, value in train_dict.items():
            self.log('train/'+name, float(value), prog_bar=True, sync_dist=True)

        return {'loss':train_loss}

    def training_epoch_end(self, outputs):
        avg_train_loss = torch.stack(
            [x['loss'] for x in outputs]
            ).mean()

        with open(self.temp_train_pred, 'w') as save_pred, open(self.temp_train_gold, 'w') as
save_gold:
            save_pred.write(self.all_pred_train)
            save_gold.write(self.all_gold_train)

        self.list_train_loss_epoch.append(avg_train_loss)

        train_dict = {
            'epoch': self.epoch,
            'loss': avg_train_loss
            }
        for name, value in train_dict.items():
            self.log('train/'+name, float(value), prog_bar=True, sync_dist=True)

        self.avg_train_loss = avg_train_loss

        if self.hparams.no_use_token_type_ids:
            temp_token_type_ids = "no-token-type-ids"
        else:
            temp_token_type_ids = "use-token-type-ids"
        model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
        self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
            model_name_or_path, temp_token_type_ids,
            self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
            self.hparams.classifier_dropout, self.hparams.batch_size, self.epoch
            )

        ckpt_path = f"{self.dir_path}/{self.save_file}.bin"
        optimizer = self.optimizers()
        optimizer = optimizer.optimizer

        train_histories = {
            'loss':self.list_train_loss_epoch,
            }
        val_histories = {
            'loss':self.list_val_loss_epoch,
```

```
        }

    saved_checkpoint = {
        'epoch': self.epoch,
        'hyperparameters': self.hparams,
        'model_state_dict': self.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'train_histories': train_histories,
        'val_histories': val_histories,
        }
    os.makedirs(f"{self.dir_path}",exist_ok=True)
    logger.info("Currently saving model in epoch %s", str(self.epoch))
    torch.save(saved_checkpoint, ckpt_path)

    best_path = "best_bert_model_checkpoints"
    best_name = ""
    best_checkpoint = {}

    # Save best model checkpoint if current validation loss is lower than minimum loss of
previous epoch
    if not self.min_loss:
        self.min_loss = self.avg_val_loss
        best_name = self.save_file
        best_checkpoint = saved_checkpoint
    else:
        if self.avg_val_loss < self.min_loss:
            self.min_loss = self.avg_val_loss
            best_name = self.save_file
            best_checkpoint = saved_checkpoint
            os.makedirs(best_path,exist_ok=True)
            logger.info("Currently saving best bert model in epoch %s", str(self.epoch))
            best_ckpt_path = f"{best_path}/{best_name}.bin"
            torch.save(best_checkpoint, best_ckpt_path)

    self.all_pred_train = ""
    self.all_gold_train = ""
    logger.info("Epoch %2d | Min val loss: %f | Current val loss: %f" %(self.epoch,
self.min_loss, self.avg_val_loss))
    self.epoch += 1


  def validation_step(self, batch, batch_idx):
    # Get batch information
    labels = batch["labels"]
    sources = batch["source"]

    # Delete labels, source, target so now batch contains everything to be inputted into the
model
    del batch["labels"]
```

```
del batch["source"]
del batch["target"]

# Compute model forward
outputs, mask = self.forward(**batch)

# Compute loss
val_loss = self.compute_loss(outputs, labels, mask)
outputs = torch.sigmoid(outputs)

# For compute F1 ROUGE score
system_summaries = []
ref_summaries = []
for idx, label in enumerate(labels):
   temp = ""
   for idy, l in enumerate(label):
      if l:
         temp += sources[idx][idy]
         temp += '<q>'
   temp = temp[:-3]
   ref_summaries.append(temp)
source_ids = (
   torch.argsort(outputs, dim=1, descending=True)
)
for idx, (source, source_ids, target) in enumerate(
   zip(sources, source_ids, ref_summaries)
):
   pos = []
   for sent_idx, i in enumerate(source_ids):
      if i >= len(source):
         continue

      pos.append(i.item())

      if len(pos) == self.hparams.test_k:
         break
   pos.sort()
   selected_sentences = "<q>".join([source[i] for i in pos])
   system_summaries.append(selected_sentences)


if self.hparams.no_use_token_type_ids:
   temp_token_type_ids = "no-token-type-ids"
else:
   temp_token_type_ids = "use-token-type-ids"
model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
   model_name_or_path, temp_token_type_ids,
```

```
        self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
        self.hparams.classifier_dropout, self.hparams.batch_size, self.hparams.max_epochs
        )

    self.temp_val_gold = self.save_file + "/val_gold_"+ str(self.epoch) +".txt"
    self.temp_val_pred = self.save_file + "/val_pred_"+ str(self.epoch) +".txt"

    os.makedirs(self.save_file, exist_ok=True)

    for pred in system_summaries:
      self.all_pred_val += str(pred).strip() + "\n"
    for gold in ref_summaries:
      self.all_gold_val += str(gold).strip() + "\n"

    val_dict = {
      'epoch': self.epoch,
      'loss': val_loss
      }

    for name, value in val_dict.items():
      self.log('val/'+name, float(value), prog_bar=True, sync_dist=True)

    return {'loss':val_loss}

  def validation_epoch_end(self, outputs):
    # Get the average loss over all evaluation runs
    avg_val_loss = torch.stack(
      [x['loss'] for x in outputs]
    ).mean()

    with open(self.temp_val_pred, 'w') as save_pred, open(self.temp_val_gold, 'w') as
save_gold:
      save_pred.write(self.all_pred_val)
      save_gold.write(self.all_gold_val)

    self.avg_val_loss = avg_val_loss
    self.list_val_loss_epoch.append(avg_val_loss)

    val_dict = {
      'epoch': self.epoch,
      'loss': avg_val_loss,
      }

    for name, value in val_dict.items():
      self.log('val/'+name, float(value), prog_bar=True, sync_dist=True)

    self.all_pred_val = ""
    self.all_gold_val = ""
```

```
  def test_step(self, batch, batch_idx):
    # Get batch information
    labels = batch["labels"]
    sources = batch["source"]
    targets = batch["target"]

    # Delete labels, source, and target so now batch contains everything to be inputted into
the model
    del batch["labels"]
    del batch["source"]
    del batch["target"]

    # Compute model forward
    outputs, _ = self.forward(**batch)
    outputs = torch.sigmoid(outputs)

    sorted_ids = (
      torch.argsort(outputs, dim=1, descending=True).detach().cpu().numpy()
    )

    predictions = []

    if self.ref_summary:
      ref_summaries = targets
    else:
      ref_summaries = []
      for idx, label in enumerate(labels):
        temp = ""
        for idy, l in enumerate(label):
          if l:
            temp += sources[idx][idy]
            temp += '<q>'
        temp = temp[:-3]
        ref_summaries.append(temp)


    # Get ROUGE scores for each (source, target) pair
    for idx, (source, source_ids, target) in enumerate(
      zip(sources, sorted_ids, ref_summaries)
    ):
      current_prediction = []
      pos = []
      for sent_idx, i in enumerate(source_ids):
        if i >= len(source):
          continue

        candidate = source[i].strip()
```

```
            if (not self.hparams.no_test_block_ngrams) and (
                not block_ngrams(candidate, current_prediction, self.hparams.n_gram_blocking)
            ):
                current_prediction.append(candidate)
                pos.append(i.item())

            if len(current_prediction) == self.hparams.test_k:
                break
        pos.sort()
        current_prediction = "<q>".join([source[i] for i in pos])
        predictions.append(current_prediction)

    if self.hparams.no_use_token_type_ids:
        temp_token_type_ids = "no-token-type-ids"
    else:
        temp_token_type_ids = "use-token-type-ids"
    model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
    self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
        model_name_or_path, temp_token_type_ids,
        self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
        self.hparams.classifier_dropout, self.hparams.batch_size, self.epoch
        )
    self.temp_test_gold = self.save_path_test + "/" + self.save_file + "_test_gold.txt"
    self.temp_test_pred = self.save_path_test + "/" + self.save_file + "_test_pred.txt"

    # Gather all summaries in single text file
    os.makedirs(self.save_path_test, exist_ok=True)

    for pred in predictions:
        self.all_pred_test += str(pred).strip() + "\n"
    for gold in ref_summaries:
        self.all_gold_test += str(gold).strip() + "\n"

    return None

  def test_epoch_end(self, outputs):
    with open(self.temp_test_pred, 'w') as save_pred, open(self.temp_test_gold, 'w') as
save_gold:
        save_pred.write(self.all_pred_test)
        save_gold.write(self.all_gold_test)

    # ROUGE scoring
    raw_rouge, rouge_score = compute_rouge_score(
        self.fold, self.epoch, self.save_file, self.temp_test_dir, self.temp_test_pred,
self.temp_test_gold
        )
    results_dir = "results"
```

```
    os.makedirs(results_dir,exist_ok=True)
    type_ref = "_abstractive" if self.ref_summary else "_extractive"
    file = results_dir + "/" + self.save_file + type_ref + ".txt"
    with open(file,'w') as f:
      f.write("Precision\n")
      f.write(
        "ROUGE-1 : {} \nROUGE-2 : {} \nROUGE-L : {}\n\n".format(
          rouge_score['precision-rouge-1'],
          rouge_score['precision-rouge-2'],
          rouge_score['precision-rouge-l']
          )
        )
      f.write("Recall\n")
      f.write(
        "ROUGE-1 : {} \nROUGE-2 : {} \nROUGE-L : {}\n\n".format(
          rouge_score['recall-rouge-1'],
          rouge_score['recall-rouge-2'],
          rouge_score['recall-rouge-l']
          )
        )
      f.write("F1-Score\n")
      f.write(
        "ROUGE-1 : {} \nROUGE-2 : {} \nROUGE-L : {}\n\n".format(
          rouge_score['f1-rouge-1'],
          rouge_score['f1-rouge-2'],
          rouge_score['f1-rouge-l']
          )
        )


    test_dict = {
      **rouge_score,
      **raw_rouge
      }

    # Generate logs
    for name, value in test_dict.items():
      self.log('test/'+name, float(value), prog_bar=True, sync_dist=True)


    self.test_preds = []
    self.test_labels = []
    self.all_pred_test = ""
    self.all_gold_test = ""
```

4.7 Transformer encoder

Pada bagian ini akan dilakukan klasifikasi dengan menggunakan encoder dari transformer. Tujuan dari penggunaan *transformer encoder* ini pada layer klasifikasi adalah untuk mempelajari relasi antar kalimat dari output yang dihasilkan oleh model BERT. *Position encoding* yang digunakan pada model BERT akan dipelajari secara otomatis saat melakukan training sedangkan *position encoding* yang dipakai sebagai *classifier* pada tambahan *transformer encoder layer* akan menggunakan *sinusoidal position encoding*. Penggunaan dari *position encoding* bertujuan untuk memberikan informasi mengenai posisi tiap *token* dalam kalimat. Mengenai proses dalam melakukan klasifikasi dapat dilihat pada Segmen Program 4.10.

Segmen Program 4.10 Implementasi klasifikasi pada transformer encoder

```
import logging
import torch
from torch import nn
import math

logger = logging.getLogger(__name__)

# Sinusoidal position encoding
class PositionalEncoding(nn.Module):

   def __init__(self, d_model=768, dropout=0.1, max_len=512):
      super(PositionalEncoding, self).__init__()
      self.dropout = nn.Dropout(p=dropout)
      pe = torch.zeros(max_len, d_model)
      position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
      div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) /
d_model))
      pe[:, 0::2] = torch.sin(position * div_term)
      pe[:, 1::2] = torch.cos(position * div_term)
      pe = pe.unsqueeze(0).transpose(0, 1)
      self.register_buffer('pe', pe)

   def forward(self, x):
      x = x + self.pe[:x.size(0), :]
      return self.dropout(x)

# Transformer encoder with sinusoidal position encoding
class TransformerEncoderClassifier(nn.Module):
   def __init__(
      self,
      d_model=768,
      nhead=8,
      dim_feedforward=2048,
```

```
        dropout=0.1,
        num_layers=2
    ):
        super(TransformerEncoderClassifier, self).__init__()
        self.nhead = nhead
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        encoder_layer = nn.TransformerEncoderLayer(
            d_model, nhead, dim_feedforward=dim_feedforward, dropout=dropout
        )
        layer_norm = nn.LayerNorm(d_model)
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers, norm=layer_norm)
        wo = nn.Linear(d_model, 1, bias=True)
        self.reduction = wo

    def forward(self, x, mask):

        # apply sinusoidal position encoding to input sequence of token
        x = self.pos_encoder(x)

        # add dimension in the middle
        attn_mask = mask.unsqueeze(1)

        # expand the middle dimension to the same size as the last dimension (the number of
sentences/source length)
        attn_mask = attn_mask.expand(-1, attn_mask.size(2), -1)

        # repeat the mask for each attention head
        attn_mask = attn_mask.repeat(self.nhead, 1, 1)

        # attn_mask is shape (batch size*num_heads, target sequence length, source sequence
length)
        # set all the 0's (False) to negative infinity and the 1's (True) to 0.0 because the
attn_mask is additive
        attn_mask = (
            attn_mask.float()
            .masked_fill(attn_mask == 0, float("-inf"))
            .masked_fill(attn_mask == 1, float(0.0))
        )

        x = x.transpose(0, 1)
        # x is shape (source sequence length, batch size, feature number)

        x = self.encoder(x, mask=attn_mask)
        # x is still shape (source sequence length, batch size, feature number)

        x = x.transpose(0, 1).squeeze()
        # x is shape (batch size, source sequence length, feature number)

        x = self.reduction(x)
```

```
    # x is shape (batch size, source sequence length, 1)
    # mask is shape (batch size, source sequence length)
    sent_scores = x.squeeze(-1) * mask.float()

    # to preserve loss calculation on padding token
    sent_scores[sent_scores == 0] = -9e3

    return sent_scores
```

4.8 Evaluasi dengan ROUGE

Dalam mengukur model yang sudah dibuat, akan dilakukan evaluasi terhadap dataset *test* dengan menggunakan ROUGE score. Setiap pasangan kandidat kalimat yang akan menjadi ringkasan sistem dengan referensi ringkasan akan disimpan ke dalam file teks yang kemudian akan dilakukan evaluasi dengan ROUGE. Implementasi untuk evaluasi ini dapat dilihat pada Segmen Program 4.11.

Segmen Program 4.11 Evaluasi ROUGE

```
def compute_rouge_score(fold, epoch, save_file, temp_dir, cand, ref):
   candidates = [line.strip() for line in open(cand, encoding="utf-8")]
   references = [line.strip() for line in open(ref, encoding="utf-8")]
   assert len(candidates) == len(references)

   cnt = len(candidates)
   current_time = time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime())
   os.makedirs(temp_dir, exist_ok=True)
   tmp_dir = os.path.join(temp_dir, "rouge-score-{}".format(current_time))
   os.makedirs(tmp_dir, exist_ok=True)
   os.makedirs(f"{tmp_dir}/candidate_{fold}", exist_ok=True)
   os.makedirs(f"{tmp_dir}/reference_{fold}", exist_ok=True)

   try:

      for i in range(cnt):
         if len(references[i]) < 1:
            continue
         with open(
            f"{tmp_dir}/candidate_{fold}/cand.{i}.txt", "w", encoding="utf-8"
         ) as f:
            f.write(candidates[i].replace("<q>", "\n"))
         with open(
            f"{tmp_dir}/reference_{fold}/ref.{i}.txt", "w", encoding="utf-8"
         ) as f:
            f.write(references[i].replace("<q>", "\n"))
```

```
    # pyrouge
    r = pyrouge.Rouge155()
    r.model_dir = f"{tmp_dir}/reference_{fold}/"
    r.system_dir = f"{tmp_dir}/candidate_{fold}/"
    r.model_filename_pattern = "ref.#ID#.txt"
    r.system_filename_pattern = "cand.(\d+).txt"
    command = '-e /content/rouge/tools/ROUGE-1.5.5/data -a -b 75 -c 95 -m -n 2'
    rouge_results = r.convert_and_evaluate(rouge_args=command)
    results_dict_rouge = r.output_to_dict(rouge_results)

    rouge_1_recall = "{:.2f}".format(float(results_dict_rouge['rouge_1_recall'] * 100))
    rouge_2_recall = "{:.2f}".format(float(results_dict_rouge['rouge_2_recall'] * 100))
    rouge_l_recall = "{:.2f}".format(float(results_dict_rouge['rouge_l_recall'] * 100))

    rouge_1_precision = "{:.2f}".format(float(results_dict_rouge['rouge_1_precision'] *
100))
    rouge_2_precision = "{:.2f}".format(float(results_dict_rouge['rouge_2_precision'] *
100))
    rouge_l_precision = "{:.2f}".format(float(results_dict_rouge['rouge_l_precision'] * 100))

    rouge_1_f_score = "{:.2f}".format(float(results_dict_rouge['rouge_1_f_score'] * 100))
    rouge_2_f_score = "{:.2f}".format(float(results_dict_rouge['rouge_2_f_score'] * 100))
    rouge_l_f_score = "{:.2f}".format(float(results_dict_rouge['rouge_l_f_score'] * 100))
    results = {}

    # Recall
    results['recall-rouge-1'] = rouge_1_recall
    results['recall-rouge-2'] = rouge_2_recall
    results['recall-rouge-l'] = rouge_l_recall

    # Precision
    results['precision-rouge-1'] = rouge_1_precision
    results['precision-rouge-2'] = rouge_2_precision
    results['precision-rouge-l'] = rouge_l_precision

    # F1-Score
    results['f1-rouge-1'] = rouge_1_f_score
    results['f1-rouge-2'] = rouge_2_f_score
    results['f1-rouge-l'] = rouge_l_f_score



    # save rouge score at specified file
    corpus_type = str(tmp_dir.split('_')[0])
    evaluation_dir = "final_evaluation_" + corpus_type
    os.makedirs(evaluation_dir, exist_ok=True)
    with open(os.path.join(evaluation_dir, f"{save_file}.txt"), 'w', encoding='utf-8') as f:
      f.write("ROUGE Score\n")
      f.write(rouge_results + '\n\n')
  finally:
```

```
    if os.path.isdir(tmp_dir):
        shutil.rmtree(tmp_dir)

  return results_dict_rouge, results
```

4.9 Prediksi ringkasan

Dalam melakukan prediksi suatu ringkasan, akan dilakukan *load* dari *checkpoint model*. Input teks artikel berita yang dimasukkan ke dalam model akan dilakukan pemotongan apabila melebihi *maximum length* dari model BERT. Mengenai implementasi dari prediksi ringkasan dapat dilihat pada Segmen Program 4.12.

Segmen Program 4.12 Prediksi ringkasan berita

```
import os
import sys
import requests
import torch
from argparse import ArgumentParser
from newspaper import Article

sys.path.insert(0, os.path.abspath("./src"))
from extractive import ExtractiveSummarizer

parser = ArgumentParser(description='Indonesian News Summarization')
parser.add_argument('--model', type=str, default='./models/indolem-indobert-base-
uncased_basic_use-token-type-ids_sent-rep-tokens_transformer_position_2_2_3e-
05_0.3_8_1.bin',help='trained model')
parser.add_argument('--source',type=str, default='./kumpulan_berita/berita_4_cnn.txt',
help='source of news')
parser.add_argument('--percentages',type=float, default='20.0',help='percentages of
summary sentences')
parser.add_argument('--save_dir', type=str, default='./kumpulan_berita', help='directory of
saved news article')

args = parser.parse_args()

checkpoint = torch.load(args.model, map_location=torch.device('cpu'))
state_dict = checkpoint['model_state_dict']
model = ExtractiveSummarizer(checkpoint['hyperparameters'])
model.load_state_dict(checkpoint['model_state_dict'])


# Check source
try:
  #Get news article
  assert(requests.get(args.source))
```

```
    news_article = Article(args.source)
    news_article.download()
    news_article.parse()
    temp = news_article.text.split('\n\n')
    contents = [sent+"\n" for sent in temp]
    file_name = news_article.title.replace(' ','-') + '.txt'
    source = 'url/' + file_name
    with open(os.path.join(args.save_dir, file_name), 'w', encoding='utf-8') as f:
      f.write('\n'.join(contents))
except:
    # Open text file
    with open(args.source) as f:
      contents = f.readlines()
    source = 'file/' + os.path.basename(args.source)

# Convert percentages to number of sentences (based on number of sentences in news text)
num_sentences = int((args.percentages*0.01) * len(contents))

# Check if sentences less than 1, then set number of sentence to min. 1
if num_sentences < 1:
        num_sentences = 1

# Predict
output = model.predict(contents, source, num_sentences)
print(output)
```

5. PENGUJIAN SISTEM

Pada bab ini akan dibahas tentang pengujian yang dilakukan pada model *Bidirectional Encoder Representations from Transformers* yang telah dibuat dan dilakukan pembahasan dari pengujian yang dilakukan. Pengujian juga dilakukan untuk mencari tahu konfigurasi yang tepat dengan melakukan percobaan *training*. Pada penelitian ini, pengujian awal dan konfigurasi terbaik dari model akan diimplementasikan pada *5 fold*. Tiap pengujian akan dibandingkan terhadap referensi ekstraktif dan abstraktif. Beberapa pengujian hanya akan dilakukan pada 1 *fold* saja untuk mengalokasikan waktu pada pengujian-pengujian yang lainnya. Selain itu, karena adanya keterbatasan *resource* yang dimiliki oleh peneliti, maka pengujian tidak sepenuhnya menggunakan konfigurasi *hyperparameter* dari penelitian sebelumnya yang telah dicoba pada berita berbahasa Inggris (Liu & Lapata, 2019). Pada penelitian ini akan dicari tahu konfigurasi *hyperparameter* yang optimal pada *resource* yang disediakan oleh *google colabarotary*. Dataset yang dipakai pada penelitian ini berasal dari penelitian yang telah dilakukan oleh Kurniawan dan Louvan dalam membangun dataset untuk mendukung penelitian pembuatan ringkasan otomatis pada berita berbahasa Indonesia yang diberi nama Indosum (Kurniawan & Louvan, 2018). Jumlah kalimat yang dijadikan ringkasan yaitu sebanyak 3 kalimat untuk semua pengujian. Hanya diambil sebanyak 3 kalimat karena untuk menyamakan dengan pengujian dari metode-metode yang digunakan oleh Kurniawan dan Louvan yaitu menggunakan 3 kalimat. Penilaian pengujian yang dilakukan pada penelitian ini akan menggunakan *ROUGE score* mengikuti penilaian standar yang digunakan pada penelitian-penelitian lain yang serupa. Nilai *ROUGE score* yang akan diambil pada penelitian ini adalah *F1-Score* dari *ROUGE-1*, *ROUGE-2*, dan *ROUGE-L*.

5.1 Pengujian Model

Pengujian model *Bidirectional Encoder Representations from Transformers* akan dilakukan dengan cara melakukan konfigurasi terhadap *hyperparameter*. Jumlah distribusi data *training*, *dev*, dan *test* untuk masing-masing *fold* dapat dilihat pada Tabel 5.1. Pembagian data *training*, *dev*, dan *test* pada penelitian ini akan disamakan dengan percobaan yang dilakukan oleh Kurniawan dan Louvan (Kurniawan & Louvan, 2018).

Tabel 5.1 Pembagian data tiap *fold*

Fold	Train	Dev	Test
1	14262	750	3762
2	14263	749	3762
3	14290	747	3737
4	14272	750	3752
5	14266	747	3761

Model yang digunakan pada pengujian ini adalah indolem/indobert-base-uncased dan bert-base-multilingual-uncased. Alasan pemilihan model ini adalah karena model indolem/indobert-base-uncased memiliki banyak kumpulan kata-kata yang terdapat dalam berita berbahasa Indonesia sedangkan untuk model bert-base-multilingual-uncased dipilih karena dapat digunakan untuk banyak bahasa termasuk bahasa Indonesia. *Hyperparameter* konstan yang digunakan pada penelitian ini dapat dilihat pada Tabel 5.2. Alasan penggunaan *batch size* sebesar 8 adalah karena keterbatasan memori pada *resource* yang digunakan pada *google colaboratory* serta peneliti menemukan bahwa model BERT yang menerima input dengan panjang token 450 keatas hanya dapat dilakukan dengan maksimum *batch size* sebesar 8 dan untuk panjang token dibawah 450 masih dapat dilakukan dengan menggunakan *batch size* sebesar 16 pada *resource* yang dipinjamkan oleh *google colaboratory*. Karena panjang rata-rata untuk bert-base-multilingual-uncased berada diatas 450 maka untuk seluruh pengujian yang akan dilakukan pada penelitian ini akan disamakan menggunakan *batch size* sebesar 8. Lalu, untuk keterangan konfigurasi *hyperparameter* dalam pengujian model dapat dilihat pada Tabel 5.3.

Tabel 5.2 Konfigurasi *hyperparameter* konstan

Batch size	Max Epochs	Epsilon	Weight Decay	Optimizer
8	4	1e-8	0.01	AdamW

Tabel 5.3 Keterangan konfigurasi *hyperparameter* untuk pengujian model BERT

#	Nama Konfigurasi	Keterangan
1	*Learning rate*	*Hyperparameter* yang digunakan untuk menghitung nilai bobot pada proses *training*. Bila nilai ini semakin rendah maka *training* akan bekerja semakin lama sedangkan semakin tinggi nilai *learning rate* maka semakin cepat *training* bekerja namun tidak optimal.
2	*Dropout*	Layer yang digunakan untuk mencegah model mengalami *overfitting* dengan menonaktifkan beberapa *neuron* yang dipilih secara acak berdasarkan persentase yang telah diinisialisasi.
3	*Token type ids*	Tipe token ids yang digunakan untuk membedakan antar kalimat dalam teks.
4	*Stacked Transformer Encoder*	Banyaknya susunan *transformer encoder* yang dipakai untuk melakukan klasifikasi terhadap kalimat yang akan diambil sebagai ringkasan.

5.1.1 Pengujian Awal

Pada pengujian ini akan dilakukan dengan menggunakan konfigurasi awal yang dapat dilihat pada Tabel 5.4. Untuk pengujian awal ini akan dilakukan *testing* terhadap *5 fold* dari data yang sudah disiapkan sebelumnya. Hasil *testing* pengujian awal dapat dilihat pada Tabel 5.5 dan 5.6.

Tabel 5.4 Konfigurasi yang digunakan untuk pengujian awal model BERT

Learning rate	Dropout	Stacked Transformer Encoder	Token type ids
1e-5	0.1	2	Use token type ids

Tabel 5.5 Pengujian Awal pada Referensi Ekstraktif

Fold	IndoBERT			Multilingual BERT		
	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
1	83.64	82.33	82.53	82.99	81.64	81.94
2	83.17	81.84	82.07	84.37	83.07	83.23
3	84.86	83.56	83.90	84.38	83.07	83.43
4	83.48	82.17	82.41	83.76	82.48	82.84
5	84.73	83.54	83.69	83.96	82.64	82.98
Average	83.97	82.68	82.92	83.89	82.58	82.88

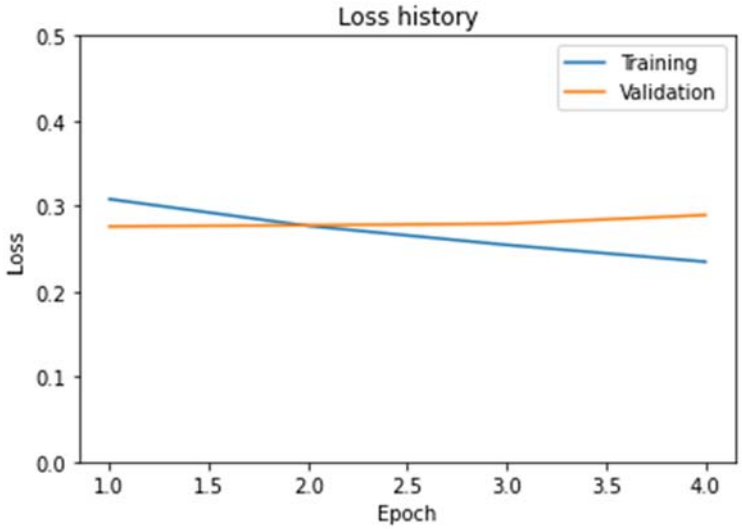
Tabel 5.6 Pengujian Awal pada Referensi Abstraktif

Fold	IndoBERT			Multilingual BERT		
	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
1	56.95	51.13	54.89	56.47	50.55	54.46
2	56.81	50.82	54.85	57.22	51.22	55.23
3	57.21	51.19	55.27	57.21	51.20	55.27
4	56.88	50.89	54.94	56.70	50.74	54.83
5	56.94	51.19	55.06	56.61	50.74	54.76
Average	56.95	51.04	55.00	56.84	50.89	54.91

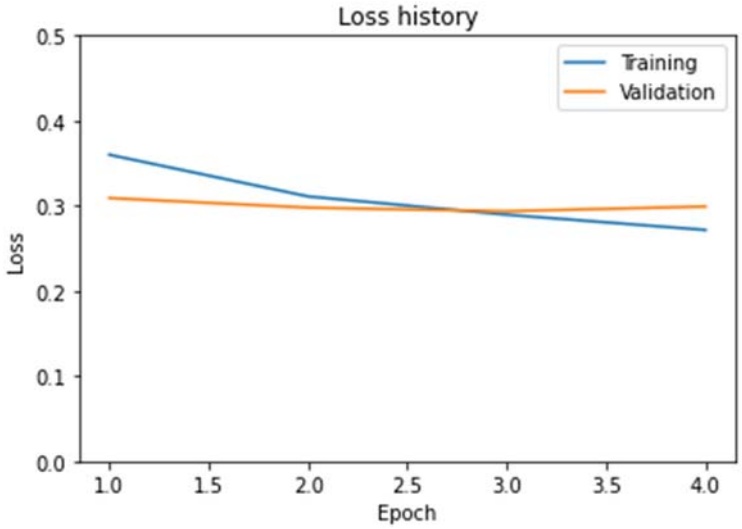
Dari pengujian awal yang telah dilakukan pada 5 *fold* data, model IndoBERT memiliki rata-rata *F1-Score* ROUGE yang lebih tinggi bila dibandingkan model Multilingual BERT. Hal ini diduga karena model IndoBERT telah dilatih sebelumnya pada korpus *wikipedia* dan berita berbahasa Indonesia oleh pembuat model sehingga ketika dilakukan percobaan menggunakan *hyperparameter* pada konfigurasi awal untuk model IndoBERT sudah dapat memperoleh hasil yang baik. Perbedaan nilai ROUGE untuk model IndoBERT dan Multilingual BERT saat diuji menggunakan referensi ekstraktif dengan referensi abstraktif masih terpaut cukup jauh yaitu sekitar 30%. Hal ini dikarenakan referensi abstraktif tidak sepenuhnya mengambil kalimat dari berita asli sehingga hasil yang didapatkan lebih rendah daripada referensi ekstraktif.

Selain itu, pada pengujian ini juga akan dilihat hasil dari training dan validasi yang bertujuan untuk mengetahui performa dari model selama proses training dan validasi berlangsung. Mengenai *loss* dari model IndoBERT yang digunakan dapat dilihat melalui grafik

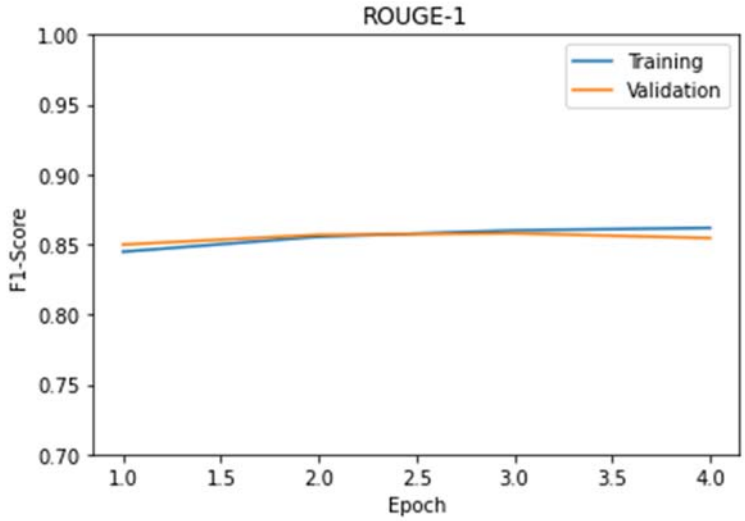
pada Gambar 5.1. *Loss* dari model Multilingual BERT dapat dilihat pada Gambar 5.2. Selain itu, akurasi *F1-Score* tiap ROUGE dari model IndoBERT yang didapatkan selama proses training dan validasi dapat dilihat pada Gambar 5.3, 5.5, dan 5.7. Akurasi *F1-Score* tiap ROUGE dari model Multilingual BERT dapat dilihat pada Gambar 5.4, 5.6, dan 5.8.



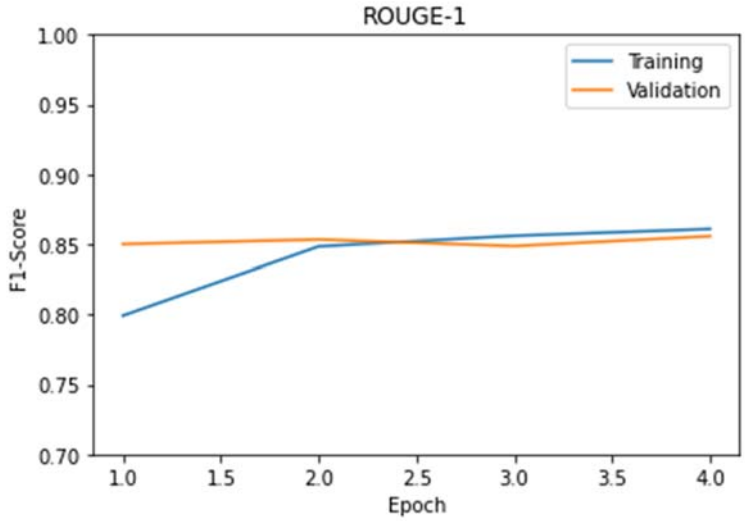
Gambar 5.1 Grafik *loss* konfigurasi awal IndoBERT



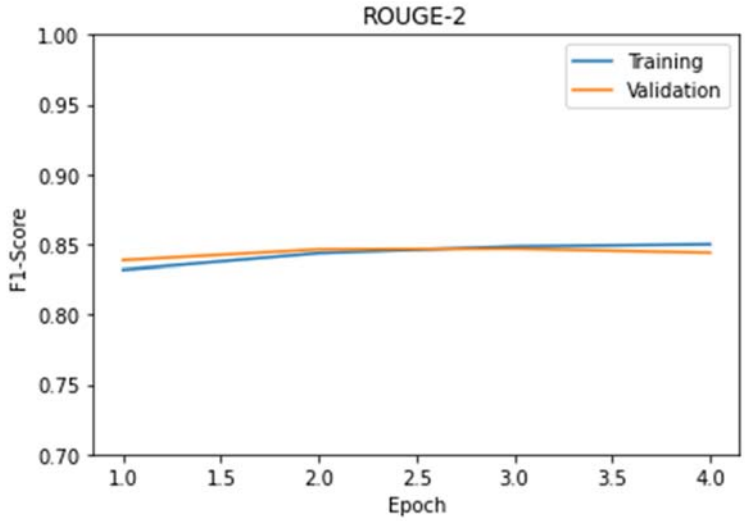
Gambar 5.2 Grafik *loss* konfigurasi awal Multilingual BERT



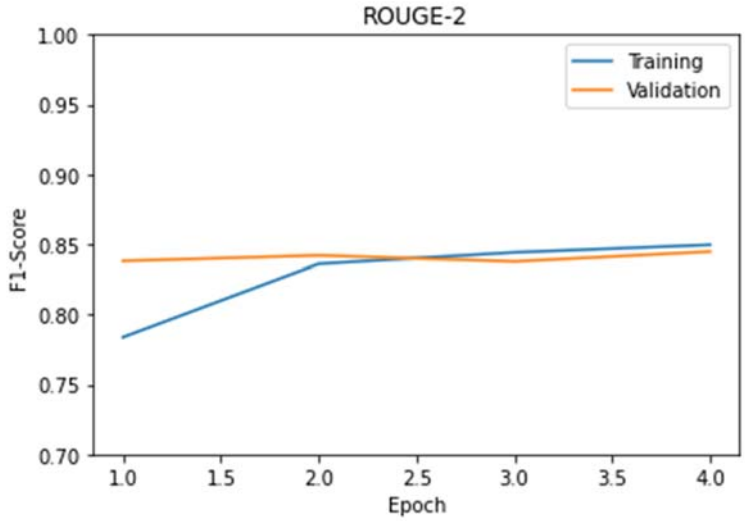
Gambar 5.3 Grafik *F1-Score* dari ROUGE-1 IndoBERT pada data *training* dan *validation*



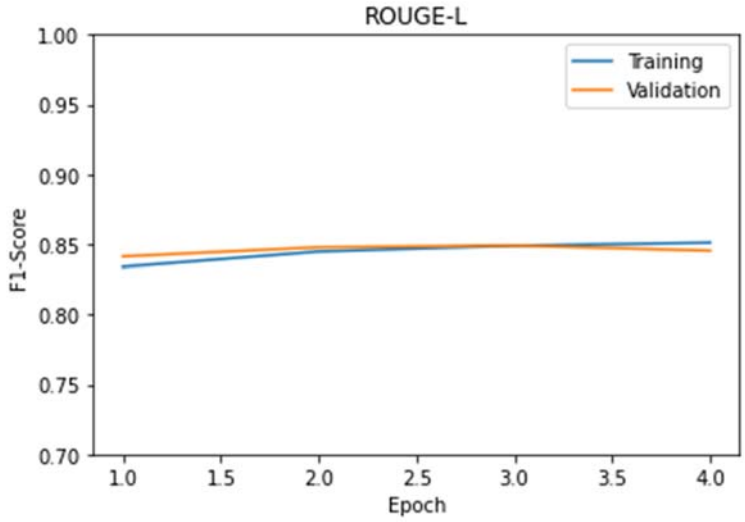
Gambar 5.4 Grafik *F1-Score* dari ROUGE-1 Multilingual BERT pada data *training* dan *validation*



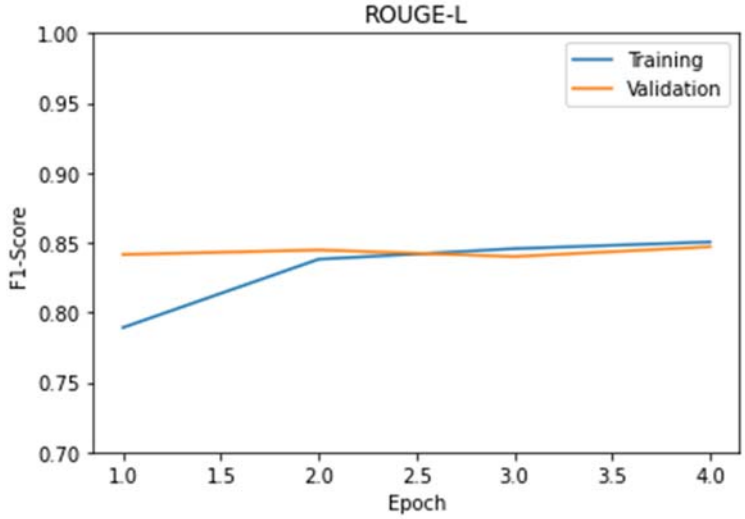
Gambar 5.5 Grafik *F1-Score* dari ROUGE-2 IndoBERT pada data *training* dan *validation*



Gambar 5.6 Grafik *F1-Score* dari ROUGE-2 Multilingual BERT pada data *training* dan *validation*



Gambar 5.7 Grafik *F1-Score* dari ROUGE-L IndoBERT pada data *training* dan *validation*



Gambar 5.8 Grafik *F1-Score* dari ROUGE-L Multilingual BERT pada data *training* dan *validation*

Berdasarkan dari grafik *loss* dan *F1-Score* ROUGE *training* dan *validation* dari model BERT, dapat diketahui bahwa untuk *validation loss* mengalami peningkatan seiring dengan bertambahnya *epoch* dan *training loss* semakin menurun yang menandakan model belajar banyak dari data *training* sehingga dapat terjadi *overfitting*. Kemudian, untuk *validation loss* dari model Multilingual BERT mengalami sedikit penurunan pada *epoch* awal. Namun, setelah *epoch* ketiga model Multilingual BERT mulai mengalami peningkatan *validation loss*. Selain itu, seiring dengan bertambahnya *epoch* untuk akurasi dari ROUGE cenderung tidak mengalami banyak perubahan pada model IndoBERT dan Multilingual BERT.

5.1.2 Pengujian Kombinasi Learning Rate dan Dropout

Pada pengujian ini akan dilakukan percobaan training dengan mengubah nilai *learning rate* dan *dropout*. Tujuan dari pengujian ini untuk mencari tahu konfigurasi *learning rate* dan *dropout* yang tepat pada model BERT yang digunakan. Selain itu, *learning rate* juga akan menentukan seberapa cepat model belajar dimana semakin tinggi nilai dari *learning rate* maka model akan belajar lebih cepat dan sebaliknya. Kemudian, *learning rate* akan dikombinasikan dengan *dropout* sehingga diharapkan dapat menghindari terjadinya *overfitting*.

Tabel 5.7 Pengujian Kombinasi Learning Rate dan Dropout pada Referensi Ekstraktif

Learning rate	Dropout	IndoBERT			Multilingual BERT		
		ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
1e-5	0.1	83.64	82.33	82.53	82.99	81.64	81.94
	0.3	84.12	82.85	83.02	83.84	82.53	82.78
	0.5	83.83	82.51	82.73	83.08	81.72	82.03
2e-5	0.1	84.11	82.84	83.00	84.02	82.77	82.95
	0.3	83.68	82.37	82.62	83.62	82.34	82.52
	0.5	84.24	83.01	83.14	82.49	81.13	81.46
3e-5	0.1	83.74	82.45	82.64	83.69	82.47	82.61
	0.3	84.50	83.22	83.40	82.19	80.76	81.18
	0.5	84.64	83.45	83.52	82.51	81.10	81.45

Tabel 5.8 Pengujian Kombinasi Learning Rate dan Dropout pada Referensi Abstraktif

Learning rate	Dropout	IndoBERT			Multilingual BERT		
		ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
1e-5	0.1	56.95	51.13	54.89	56.47	50.55	54.46
	0.3	57.21	51.40	55.12	57.06	51.26	55.03
	0.5	57.09	51.24	55.02	56.56	50.74	54.57
2e-5	0.1	57.30	51.45	55.22	57.10	51.25	55.06
	0.3	56.94	51.09	54.90	56.91	51.10	54.83
	0.5	57.38	51.56	55.29	56.03	50.17	54.05
3e-5	0.1	56.69	50.84	54.65	56.85	51.07	54.81
	0.3	57.28	51.44	55.18	55.95	50.04	53.97
	0.5	57.45	51.64	55.33	56.07	50.20	54.08

Dikarenakan *validation loss* untuk kebanyakan model IndoBERT belum ada yang mencapai nilai *loss* minimum untuk epoch diatas 1 sehingga untuk pengujian ini akan digunakan *epoch* yang pertama untuk model IndoBERT yang dilatih sebagai perbandingan dengan Multilingual BERT. Dari pengujian yang telah dilakukan, model IndoBERT dapat menghasilkan

nilai ROUGE yang baik meskipun hanya diambil dari *epoch* yang pertama saja. Pengujian terbaik adalah pada saat menggunakan nilai *learning rate* sebesar 3e-5 dan *dropout* sebesar 0.5 untuk model IndoBERT sedangkan untuk model Multilingual BERT saat menggunakan nilai *learning rate* 2e-5 dan *dropout* sebesar 0.1. Untuk model BERT yang akan dipakai sebagai konfigurasi terbaik diambil dari yang diuji dengan referensi abstraktif.

5.1.3 Pengujian Token Type Ids

Pada pengujian ini akan dicoba melatih model tanpa menggunakan *token type ids* yang bertujuan untuk membedakan antara kalimat dengan urutan ganjil dan kalimat dengan urutan genap pada teks. Pengujian ini akan dilakukan menggunakan konfigurasi *learning rate* dan *dropout* terbaik dari pengujian sebelumnya. Tujuan dilakukannya pengujian ini adalah untuk mengetahui apakah hasil ringkasan akan menjadi lebih baik tanpa menggunakan *token type ids* yang berperan sebagai segmen antar kalimat.

Tabel 5.9 Pengujian Token Type Ids pada Referensi Ekstraktif

Token type ids	**IndoBERT**			**Multilingual BERT**		
	ROUGE-1	**ROUGE-2**	**ROUGE-L**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
No token type ids	84.36	83.13	83.26	82.99	81.56	81.99
Use token type ids	84.64	83.45	83.52	84.02	82.77	82.95

Tabel 5.10 Pengujian Token Type Ids pada Referensi Abstraktif

Token type ids	**IndoBERT**			**Multilingual BERT**		
	ROUGE-1	**ROUGE-2**	**ROUGE-L**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
No token type ids	57.23	51.41	55.15	56.44	50.50	54.43
Use token type ids	57.45	51.64	55.33	57.10	51.25	55.06

Dari pengujian ini dapat dilihat bahwa untuk *token type ids* berpengaruh terhadap nilai ROUGE dimana tanpa menggunakan *token type ids* terjadi sedikit penurunan terhadap *F1-Score* ROUGE hasil ringkasan. Selain itu, pada model IndoBERT dapat memperoleh nilai ROUGE yang lebih tinggi bila dibandingkan dengan model Multilingual BERT.

5.1.4 Pengujian Stacked Transformer Encoder

Pada pengujian ini akan dilakukan dengan mengatur tumpukan *transformer encoder* yang digunakan dalam melakukan klasifikasi dari kalimat yang akan dipilih sebagai ringkasan. Pengujian ini dilakukan untuk mengetahui seberapa baik kualitas ringkasan sistem yang dihasilkan saat dilakukan penumpukan sejumlah 'n' *layer transformer encoder* saat mempelajari hubungan antar kalimat yang akan dipilih sebagai ringkasan. Konfigurasi untuk *learning rate* dan *dropout* pada pengujian ini menggunakan *learning rate* dan *dropout* terbaik pada pengujian sebelumnya.

Tabel 5.11 Pengujian *Stacked Transformer Encoder* pada Referensi Ekstraktif

N-Stacked Transformer Encoder	IndoBERT			Multilingual BERT		
	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
1	83.89	82.60	82.75	84.14	82.92	83.11
2	84.64	83.45	83.52	84.02	82.77	82.95
3	84.43	83.16	83.36	83.61	82.34	82.59

Tabel 5.12 Pengujian *Stacked Transformer Encoder* pada Referensi Abstraktif

N-Stacked Transformer Encoder	IndoBERT			Multilingual BERT		
	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
1	57.08	51.23	54.99	57.17	51.36	55.14
2	57.45	51.64	55.33	57.10	51.25	55.06
3	57.13	51.28	55.07	56.86	51.06	54.84

Dari pengujian ini dapat diketahui bahwa jumlah layer *transformer encoder* mempengaruhi nilai ROUGE namun tidak signifikan. Kemudian, untuk model IndoBERT memiliki konfigurasi terbaik menggunakan 2 *transformer encoder layer* sedangkan untuk Multilingual BERT memiliki konfigurasi terbaik dengan menggunakan 1 *transformer encoder layer*.

5.1.5 Pengujian Akhir

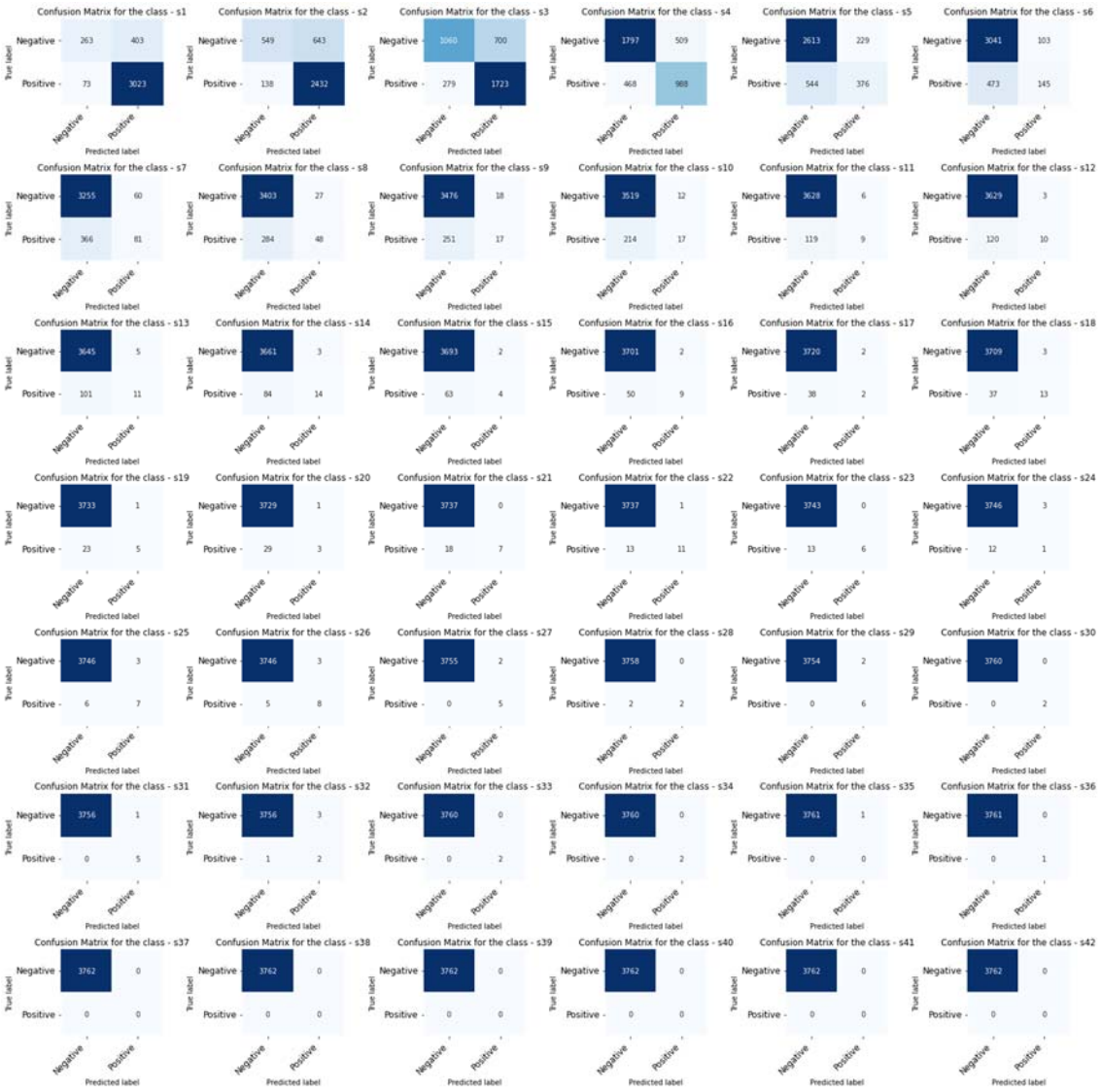
Pada pengujian akhir ini akan dilakukan dengan menggunakan konfigurasi terbaik pada 5 *fold* data. Tujuan dari pengujian ini untuk mengetahui performa model dengan menggunakan konfigurasi terbaik yang didapat dari percobaan sebelumnya terhadap seluruh *fold*. Pengujian ini akan dilakukan pada model IndoBERT saja dikarenakan untuk nilai ROUGE terbaik dari pengujian-pengujian sebelumnya adalah pada saat menggunakan model IndoBERT.

Tabel 5.13 Pengujian Akhir pada Referensi Ekstraktif

Fold	ROUGE-1	ROUGE-2	ROUGE-L
1	84.64	83.45	83.52
2	83.68	82.31	82.60
3	85.03	83.79	84.03
4	84.31	83.08	83.25
5	84.64	83.43	83.62
Average	84.46	83.21	83.40

Tabel 5.14 Pengujian Akhir pada Referensi Abstraktif

Fold	ROUGE-1	ROUGE-2	ROUGE-L
1	57.45	51.64	55.33
2	56.92	50.90	54.97
3	57.37	51.46	55.39
4	57.16	51.21	55.24
5	56.96	51.18	55.07
Average	57.17	51.27	55.20



Gambar 5.9 *Confusion Matrix* Model Terbaik

Berdasarkan dari *confusion matrix* pada hasil akhir model BERT terbaik didapatkan bahwa model cukup baik dalam melakukan prediksi pada kalimat-kalimat awal sebagai ringkasan berita. Namun, mulai dari kalimat ketujuh hingga akhir cenderung tidak banyak yang diprediksikan dengan positif dan mulai banyak diprediksikan dengan negatif. Hal ini menunjukkan bahwa model BERT yang telah dibuat masih belum banyak mengambil kalimat yang ada di tengah dan akhir sebagai hasil ringkasan berita.

Selain itu, metode ini juga akan dibandingkan dengan metode-metode lain yang telah diterapkan sebelumnya sebagai perbandingan. Perbandingan dari metode yang digunakan pada penelitian ini dengan metode-metode lain dapat dilihat pada Tabel 5.15.

Tabel 5.15 Perbandingan dengan metode-metode *neural network* yang sudah pernah diterapkan

Metode	ROUGE-1	ROUGE-2	ROUGE-L
NeuralSum 300 emb. size (Kurniawan & Louvan, 2018)	67.96	61.16	67.24
Bidirectional GRU-RNN (Halim et al., 2020)	57.01	51.17	55.10
BERT	57.17	51.27	55.20

Melalui pengujian yang telah dilakukan, dapat dilihat bahwa metode BERT yang digunakan pada penelitian ini memiliki ROUGE score yang masih rendah. Hal ini diduga karena adanya beberapa faktor yang dapat mempengaruhi model yang telah dibuat contohnya seperti *batch size* dimana pada penelitian sebelumnya yang dicoba pada berita berbahasa Inggris digunakan *batch size* yang sangat tinggi untuk mendapatkan nilai ROUGE score yang tinggi dari model BERT. Selain itu, peneliti juga menduga bahwa penggunaan BERT untuk menghasilkan *embedding* kalimat masih belum sebaik *word2vec* dalam hal menangkap maksud dari suatu kalimat dikarenakan BERT hanya dilakukan pada level kalimat saja sehingga hasilnya masih lebih rendah bila dibandingkan dengan metode sebelumnya yang dilakukan oleh Kurniawan dan Louvan (Kurniawan & Louvan, 2018). Untuk penelitian ini digunakan *batch size* rendah untuk menyesuaikan dengan *resource* yang disediakan oleh *google colaboratory*. Dari pengujian-pengujian yang telah dilakukan dapat disimpulkan bahwa model BERT yang diterapkan perlu dilatih dengan menggunakan nilai *hyperparameter* yang tinggi pada *resource* yang lebih besar sehingga dapat memperoleh nilai ROUGE score yang lebih baik lagi serta untuk model IndoBERT perlu dicari tahu lagi nilai *hyperparameter* yang sesuai sehingga performa dari model BERT dapat ditingkatkan lebih lanjut. Selain itu, pada penelitian ini tidak meminta penilaian orang lain terhadap hasil ringkasan sistem. Hal tersebut dikarenakan akan sulit untuk dievaluasi karena tiap orang memiliki penilaian yang subjektif terhadap hasil ringkasan sehingga untuk penelitian ini akan digunakan referensi abstraktif yang telah dibuat oleh manusia untuk mewakili penilaian ini.

5.2 Pengujian program

Pada pengujian ini akan dilakukan percobaan apakah program yang dibuat dapat berjalan dengan baik. Mengenai desain program yang telah dibuat dapat dilihat pada pada subbab 3.3. Pada program yang telah dibuat, *user* dapat menginputkan sebuah file berita berisi teks dari artikel berita dan juga *user* dapat menginputkan *url* sebagai opsi tambahan bila *user* ingin

mengambil langsung dari sebuah portal berita online. Dalam mengambil berita dari sebuah link *url* akan dibantu dengan *library newspaper*. Karena untuk tiap website memiliki struktur yang berbeda-beda maka artikel yang dapat diekstraksi dengan *library newspaper* hanya dapat dilakukan pada beberapa portal berita saja. Persentase dari kalimat ringkasan akan menentukan seberapa banyak kalimat yang akan dijadikan sebagai ringkasan sistem. Semakin rendah persentase yang diinputkan maka hasil ringkasan yang dihasilkan oleh sistem juga akan semakin sedikit dan berlaku juga untuk sebaliknya. Karena model BERT memiliki maksimum *fixed length* sebesar 512 maka untuk panjang artikel yang melebihi 512 akan dilakukan *truncate* supaya input teks berita dapat dimasukkan ke dalam model yang sudah dibuat.

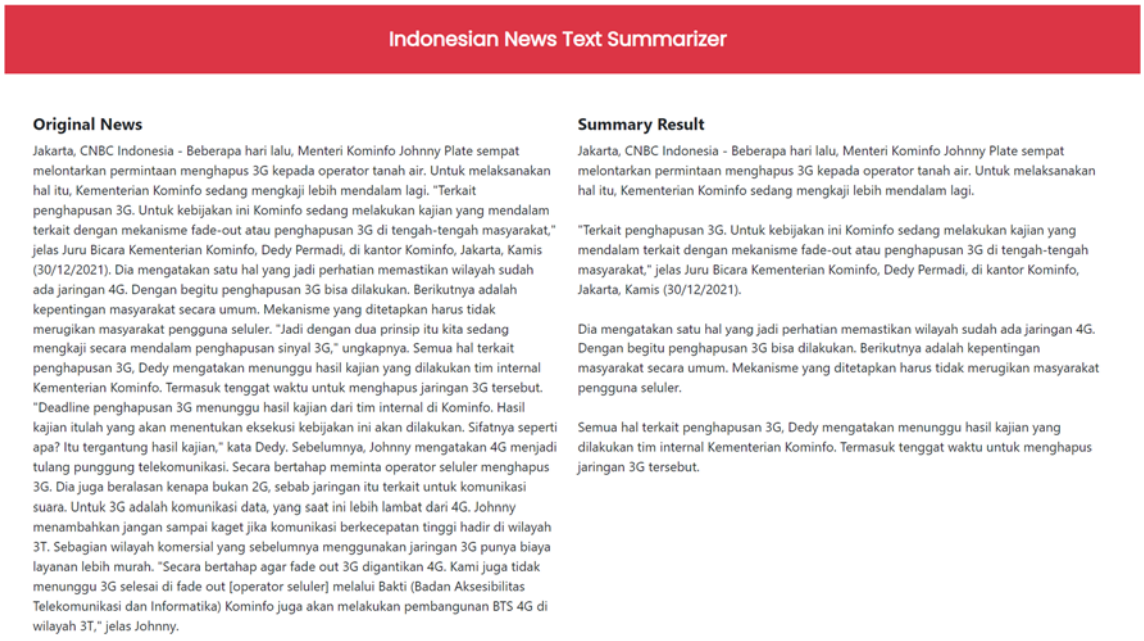
Gambar 5.10 Contoh input file teks sebuah portal berita online

Segmen Data 5.1 Isi file teks berita yang diupload

Jakarta, CNBC Indonesia - Industri penerbangan dunia harus menahan fase pemulihannya pasca pandemi Covid-19. Hal ini dikarenakan varian baru, Omicron, yang semakin meluas di beberapa negara.
Melansir laporan Reuters, setidaknya ada 4.000 penerbangan yang ditunda pada Minggu, (2/1/2022). Dari jumlah itu, sebanyak 2.400 penerbangan yang dibatalkan ada di Amerika Serikat (AS).
"Di antara maskapai dengan pembatalan terbanyak adalah SkyWest dan SouthWest dengan masing-masing 510 dan 419 pembatalan," lapor data dari aplikasi penerbangan FlightAware.
Tak hanya itu, maskapai penerbangan besar seperti Delta Air Lines juga melakukan hal yang sama. Maskapai ini membatalkan 173 penerbangan pada Malam Natal. Delta mengatakan pembatalan itu karena beberapa masalah termasuk varian Omicron.
"Kami meminta maaf kepada pelanggan kami atas keterlambatan rencana perjalanan liburan mereka," kata Delta dalam sebuah pernyataan.
"Orang-orang Delta bekerja keras untuk membawa mereka ke tempat yang mereka butuhkan secepat dan seaman mungkin pada penerbangan berikutnya yang tersedia."
Dalam data terbaru, otoritas AS mendaftarkan setidaknya 346.869 infeksi Covid-19 baru pada hari Sabtu (1/1/2022). Dari angka ini, jumlah kematian akibat Covid-19 naik setidaknya 377 kasus menjadi 828.562.
Penasehat Gedung Putih Dr Anthony Fauci sendiri sudah menghimbau agar masyarakat AS tidak melakukan perjalanan dalam sesi liburan tahun ini. Ia mengatakan bahwa Omicron telah menjadi salah satu varian paling berbahaya yang dapat menggenjot angka infeksi Covid-19 di negara itu.

Gambar 5.11 Hasil ringkasan sistem yang menggunakan input file text

Gambar 5.12 Contoh input *url* sebuah portal berita online



Gambar 5.13 Hasil ringkasan sistem dari sebuah *url* portal berita online

6. KESIMPULAN DAN SARAN

Bab ini berisi tentang kesimpulan dari ringkasan ekstraktif otomatis pada berita berbahasa Indonesia menggunakan metode Bidirectional Encoder Representations from Transformers (BERT).

6.1 Kesimpulan

Dari hasil implementasi sistem yang dilakukan, dapat ditarik beberapa kesimpulan diantaranya:

- Metode BERT yang diterapkan pada penelitian ini mendapatkan ROUGE terbaik dengan konfigurasi *learning rate* 3e-5, *dropout* 0.5, tumpukan *transformer encoder* sebanyak 2, dan menggunakan *token type ids* untuk model IndoBERT serta konfigurasi *learning rate* 2e-5, *dropout* 0.1, tumpukan *transformer encoder* sebanyak 1, dan menggunakan *token type ids* untuk model Multilingual BERT.
- Hasil ROUGE terbaik lewat pengujian yang telah dilakukan dapat menghasilkan *F1-Score* untuk ROUGE-1 sebesar 57.17, ROUGE-2 sebesar 51.27, dan ROUGE-L sebesar 55.20 pada referensi abstraktif serta ROUGE-1 sebesar 84.46, ROUGE-2 sebesar 83.21, dan ROUGE-L sebesar 83.40 pada referensi ekstraktif.
- Perbedaan nilai ROUGE antara referensi ekstraktif dan abstraktif masih cukup signifikan yaitu mendekati sekitar 30%.
- Dari berbagai pengujian yang telah dilakukan, model IndoBERT yang digunakan dapat diperbaiki lebih lanjut dengan variasi nilai *hyperparameter* lain yang lebih kecil sehingga dapat mengurangi *overfitting*.

6.2 Saran

Saran yang dapat diberikan untuk mengembangkan penelitian lebih lanjut diantaranya adalah:

- Menggunakan model transformer lain seperti OpenAI GPT, ALBERT dan BART.
- Dapat dicoba menggunakan layer klasifikasi yang lain diatas model BERT seperti LSTM.
- Dapat memperbaiki metode BERT lebih lanjut karena dari pengujian yang telah dilakukan pada penelitian ini masih terdapat beberapa kalimat yang diambil secara urut sebagai ringkasan.

DAFTAR REFERENSI

Cheng, J., & Lapata, M. (2016). Neural summarization by extracting sentences and words. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, *1*, 484–494. https://doi.org/10.18653/v1/p16-1046

Clark, K., Khandelwal, U., Levy, O., & Manning, C. D. (2019). What does BERT look at? An analysis of BERT's attention. *ArXiv*. https://doi.org/10.18653/v1/w19-4828

Deng, L., & Liu, Y. (2018). *Deep learning in natural language processing*. Springer. https://doi.org/10.1007/978-981-10-5209-5_11

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, *1*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

El-Kassas, W. S., Salama, C. R., Rafea, A. A., & Mohamed, H. K. (2020). Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, *165*, 113679. https://doi.org/10.1016/j.eswa.2020.113679

Halim, K., Novianus Palit, H., & Tjondrowiguno, A. N. (2020). Penerapan Recurrent Neural Network untuk Pembuatan Ringkasan Ekstraktif Otomatis pada Berita Berbahasa Indonesia. *Jurnal Infra*, *8*(1), 221–227.

Hirschberg, J., & Manning, C. D. (2015). Advances in Natural Language Processing. *Science*, *349*(6245), 261–266. https://doi.org/10.1126/science.aaa8685

Ismi, D. P., & Ardianto, F. (2019). Peringkasan Ekstraktif Teks Bahasa Indonesia dengan Pendekatan Unsupervised Menggunakan Metode Clustering. *CYBERNETICS*, *3*(02), 90–99.

Joshi, A., Fidalgo, E., Alegre, E., & Fernández-Robles, L. (2019). SummCoder: An unsupervised framework for extractive text summarization based on deep auto-encoders. *Expert Systems with Applications*, *129*, 200–215. https://doi.org/10.1016/j.eswa.2019.03.045

Koto, F., Rahimi, A., Lau, J. H., & Baldwin, T. (2020). IndoLEM and IndoBERT: A Benchmark Dataset and Pre-trained Language Model for Indonesian NLP. *Proceedings of the 28th International Conference on Computational Linguistics*, 757–770.

Koto, F., Lau, J. H., & Baldwin, T. (2020). *Liputan6: A Large-scale Indonesian Dataset for Text Summarization*. ArXiv. doi:arXiv:2011.00679

Kurniawan, K., & Louvan, S. (2018). Indosum: A new benchmark dataset for Indonesian text summarization. *2018 International Conference on Asian Language Processing (IALP)*, 215–220.

Lin, C. Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 74–81.

Liu, Y. (2019). Fine-tune BERT for Extractive Summarization. *ArXiv*. doi:arXiv:1903.10318v2

Liu, Y., & Lapata, M. (2019). Text summarization with pretrained encoders. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3730–3740. https://doi.org/10.18653/v1/d19-1387

Nada, A. M. A., Alajrami, E., Al-saqqa, A. A., & Abu-naser, S. S. (2020). Arabic Text Summarization Using AraBERT Model Using Extractive Text Summarization Approach. *International Journal of Academic Information System Research (IJAISR)*, *4*(8), 6–9.

Schmitt, J. B., Debbelt, C. A., & Schneider, F. M. (2017). Too much information? Predictors of information overload in the context of online news exposure. *Information Communication and Society*, *21*(8), 1151–1167. https://doi.org/10.1080/1369118X.2017.1305427

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *2017-Decem*(Nips), 5999–6009.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., & Liu, T. Y. (2020). On layer normalization in the transformer architecture. *37th International Conference on Machine Learning, ICML 2020*, *PartF16814*, 10455–10464.