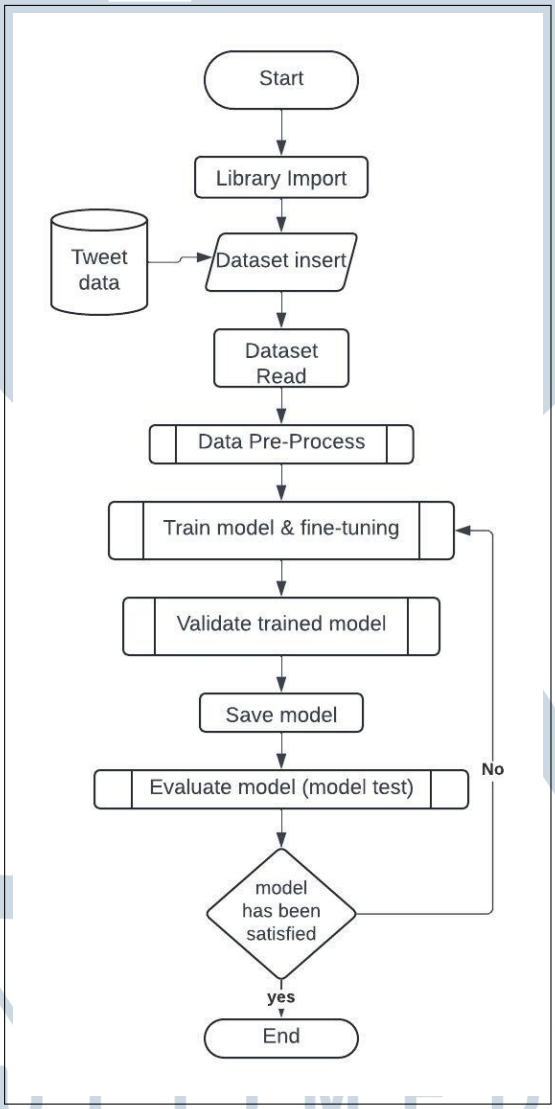
BAB 3
METODOLOGI PENELITIAN

Pada bagian ini dijabarkan langkah-langkah yang hendak dilakukan dalam menyusun dan mengerjakan penelitian. Langkah-langkah penelitian yang dijabarkan dimulai dari awal hingga akhir selesai dapat dilihat pada Gambar 3.1.



Gambar 3.1. *Workflow* Penelitian

3.1 Pengumpulan Data

Langkah pertama dalam penelitian ini adalah mengumpulkan dan membangun data yang berkaitan dengan penelitian. Data Indonesia yang akan digunakan berasal dari kaggle yang berisikan 13.168 baris kumpulan data, dengan 5.561 baris data *hate speech* dan 7.607 baris data non-*hate speech* bersumber dari Twitter, lalu data dengan bahasa Inggris terdapat 998 baris data dari penelitian yang berisikan data 433 data *hate speech* dan 565 data non-*hate speech*. Pengumpulan dan pembangunan data bilingual akan dilakukan hingga jumlah 700 data yang berasal dari Twitter, terdiri dari 232 data *hate speech* dan 468 data non-*hate speech*. Pengumpulan data dilakukan berdasarkan kalimat *hate speech* atau non-*hate speech* yang mengandung bahasa bilingual Indonesia-Inggris.

Ketiga data tersebut akan digabungkan dan dipecah secara acak menjadi *test data* dan *train data*, menggunakan *train_test_split* dengan perbandingan 90:10. Pada penelitian ini data yang akan digunakan nantinya adalah *train data*, yang kemudian akan dipecah kembali menjadi *train data* dan *validation data*. Lalu untuk evaluasi model akan digunakan 140 baris data dari *dataset* bilingual. Walaupun hasil dari *class* didalam *dataset* yang tidak seimbang, hal tersebut dilakukan demi menjaga distribusi asli dari data, tanpa melakukan duplikasi pada data.

3.2 Data Pre-Process

Pada Gambar 3.4 data yang sudah didapatkan perlu diproses sebelum dapat digunakan sebagai data latihan bagi BLOOM model, proses yang perlu dilewati adalah *tokenizing*, *padding*, dan *masking*. Langkah pertama dalam *pre-process* adalah menjatuhkan atau menghapus kolom yang tidak terpakai dari *dataset* yang asli. Setelah itu, perlu dilakukan inisialisasi BloomTokenizerFast yang akan digunakan untuk melakukan tokenisasi pada semua data. Tokenisasi akan dilakukan dengan memasukan data teks ke dalam BloomTokenizerFast yang akan menghasilkan *tokenized text*, dan Token ID yang akan digunakan untuk BLOOM Filter. Hasil dari BloomTokenizerFast adalah teks yang di tokenisasi dan Token ID (alamat *array*) yang di dapat dari teks yang di tokenisasi, seperti pada Gambar 3.2.

```
Original:  hayoo.. jokower yg pernah ngetwit bantuan sembako itu duit pribadi jkw, buru2 lah hapus twit.. :-)
Tokenized: ['h', 'ayo', 'o', '..', 'Ġjok', 'ower', 'Ġyg', 'Ġpernah', 'Ġng', 'et', 'wit', 'Ġbantuan', 'Ġsembako',
            'Ġitu', 'Ġduit', 'Ġpribadi', 'Ġj', 'kw', ',', 'Ġburu', '2', 'Ġlah', 'Ġh', 'apus', 'Ġt', 'wit', '..', 'Ġ:-', ')']
Token IDs: [75, 9151, 82, 566, 21445, 4852, 29905, 10794, 743, 364, 6251, 30040, 224495, 2071, 210683, 41132,
            504, 19760, 15, 40919, 21, 36673, 318, 21977, 261, 6251, 566, 46955, 12]
```

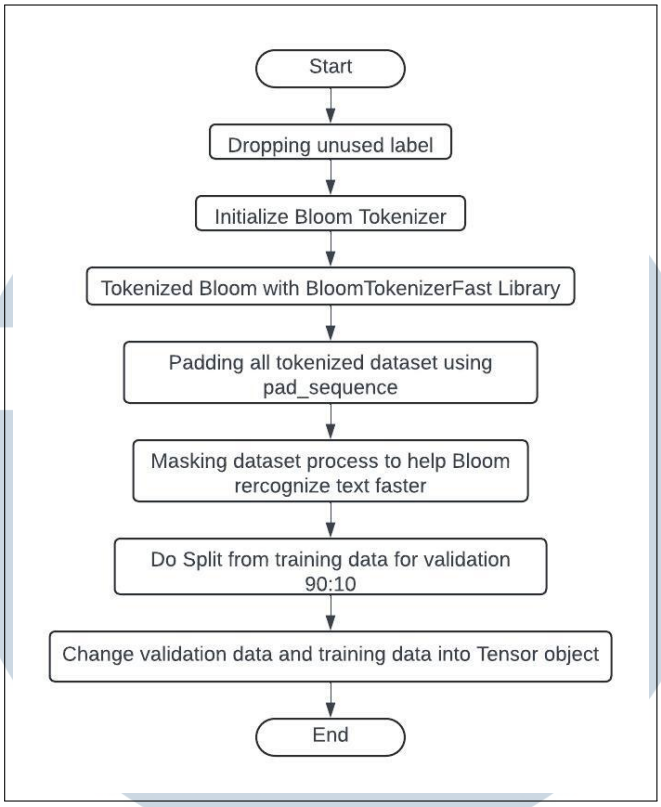
Gambar 3.2. *Tokenized text*

Data yang sudah di-tokenisasi perlu dilakukan *Padding* dengan menggunakan pad_sequence pada setiap Token ID dikarenakan panjang setiap teks yang ditokenisasi berbeda, untuk menyamakan panjang dari setiap teks ditambahkan nilai 0 dibelakang tiap Token ID sesuai dengan panjang maksimal. *Masking* dilakukan agar model dapat membedakan mana Token ID dari teks dan mana Token ID dari *Padding*. *Masking* yang dilakukan dengan membuat sebuah baris data baru yang menggambarkan letak dari setiap Token ID. Apabila Token ID memiliki nilai maka *masking* akan memberi nilai 1 pada posisi tersebut, apabila Token ID bernilai 0 maka token tersebut akan dianggap sebagai *padding* dan *masking* akan memberi nilai 0.

Contoh dari hasil *padding* dan *masking* pada *dataset* dapat dilihat pada Gambar 3.3. Hasil dari *padding* yaitu pada baris *array* ketiga yang berfungsi untuk menyamai panjang setiap teks yang sudah di tokenisasi, dan hasil dari *masking* pada baris keempat *array* yang berfungsi untuk membantu model dalam memfokuskan pada Token ID.

```
nete nya air silikon'
['n', 'ete', 'Ġnya', 'Ġair', 'Ġsil', 'ikon', "'"]
[   81  5273 28810  6738  8940 81738    10     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

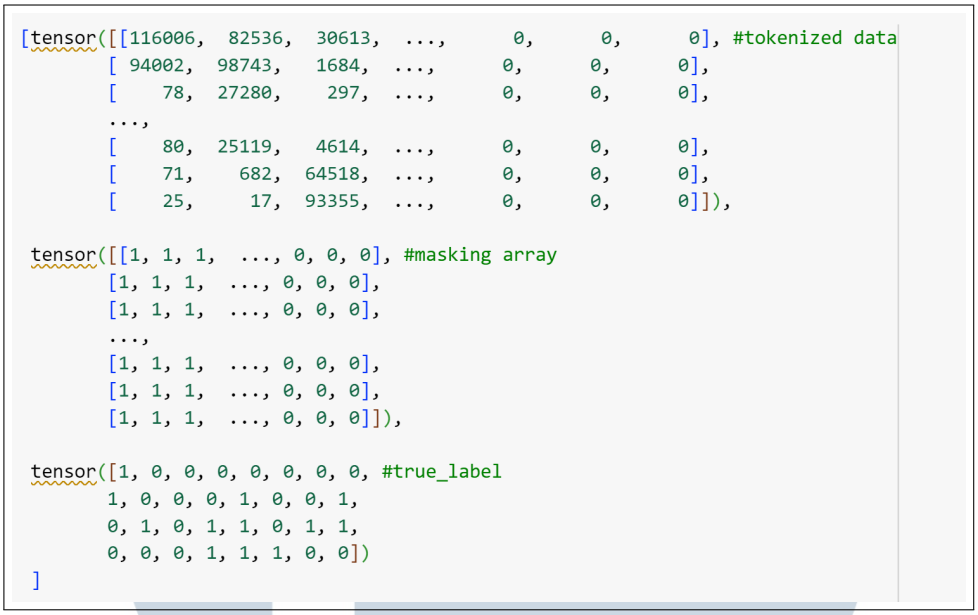
Gambar 3.3. *Padding and Masking*



Gambar 3.4. Data *Pre-process*

3.3 Model Training

Pada Gambar 3.6 data-data berupa *tokenized text* yang sudah di *padding*, *masking data* dan label data yang sudah melewati *pre-process* akan dipisah menjadi *train data* dan *validation data*, setelah dipisah data perlu diubah menjadi objek Tensor agar bisa digunakan untuk melatih BLOOM Filter model. *Train data* yang sudah dalam bentuk objek tensor akan diacak setiap *batch* nya menggunakan RandomSampler dan dipecah dengan ukuran 32 baris data per *batch* menggunakan fungsi DataLoader dari PyTorch, bentuk dari DataLoader seperti pada Gambar 3.5. Setelah *training data* disiapkan, BLOOM model akan di inisialisasi dan ditentukan berapa *epoch* model akan dilatih menggunakan *dataset*.



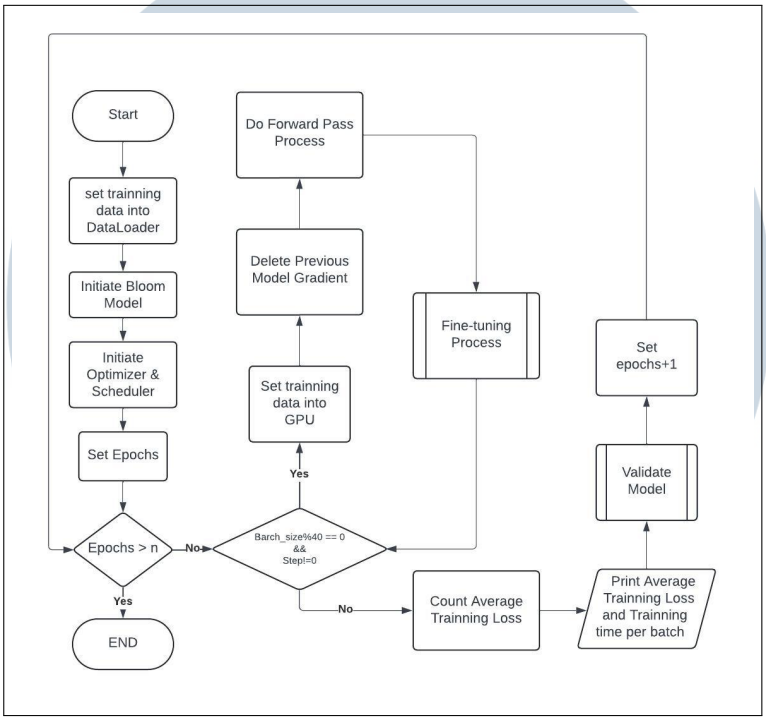
Gambar 3.5. DataLoader *output*

Model akan dilatih secara otomatis di dalam Google Colaboratory, sesuai dengan *epoch* yang sudah ditentukan. Pada setiap *epoch*, model akan dilatih menggunakan *training data* yang sudah dibagi menggunakan DataLoader, dengan memasukkan *training data* terlebih dahulu ke dalam GPU. Perhitungan gradien dari iterasi sebelumnya akan dihapus terlebih dahulu agar tidak mempengaruhi kerja dari *fine-tuning*, lalu dilakukan proses *forward pass* yaitu memasukkan data yang sudah dimasukan ke dalam GPU ke dalam model. Dari proses *forward pass* akan didapatkan hasil berupa *loss* yang masih perlu dikalkulasi dengan jumlah seluruh *training data* agar data dijadikan rata-rata *loss* pada setiap *epoch*.

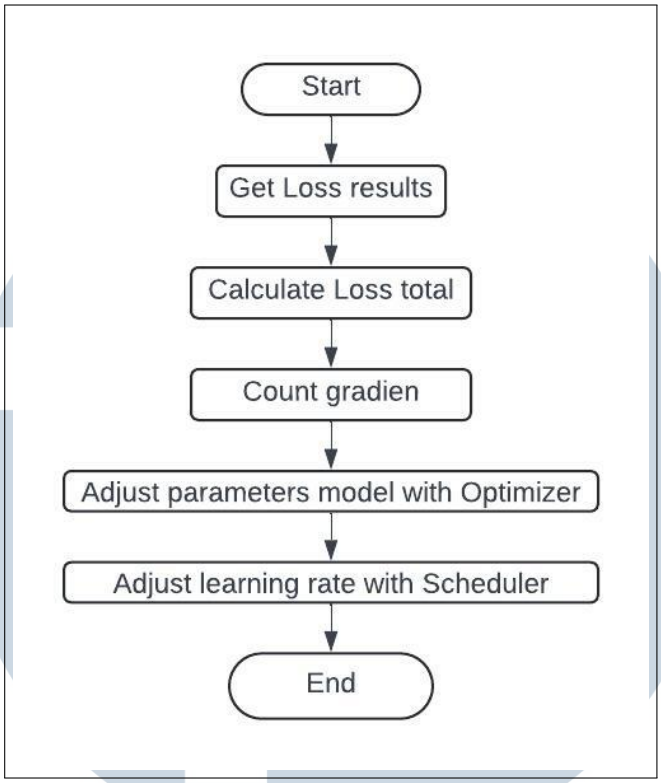
Selanjutnya akan dilakukan *backward pass* untuk menghitung gradien yang akan berguna bagi *scheduler* dan *optimizer* dalam proses *fine-tuning* model, seperti pada Gambar 3.7. Gradien yang sudah didapatkan berupa nilai *loss* yang akan digunakan oleh *optimizer* dalam optimisasi *weight* pada setiap parameter, dan akan digunakan oleh *scheduler* untuk mengatur nilai *learning rate*. Pelatihan akan dilakukan hingga memenuhi jumlah *batch* didalam satu epoch yang sudah di tentukan seperti pada Gambar 3.6.

Setelah setiap *batch* selesai dilatih ke dalam model, *epoch* akan ditambah satu lalu akan dilakukan validasi model untuk melihat akurasi model pada setiap *epoch*. Pada penelitian ini akan digunakan *optimizer* adamW. Dikarenakan BloomForSequenceClassification merupakan BLOOM Filter yang ditambahkan lapisan-lapisan pemrosesan, maka *optimizer* akan bekerja untuk mengubah

parameter tersebut di dalam BLOOM Filter secara stokastik. Dengan adanya *optimizer* dapat mempercepat dalam menyentuh nilai *loss* sekecil mungkin, tanpa banyak *epoch*.

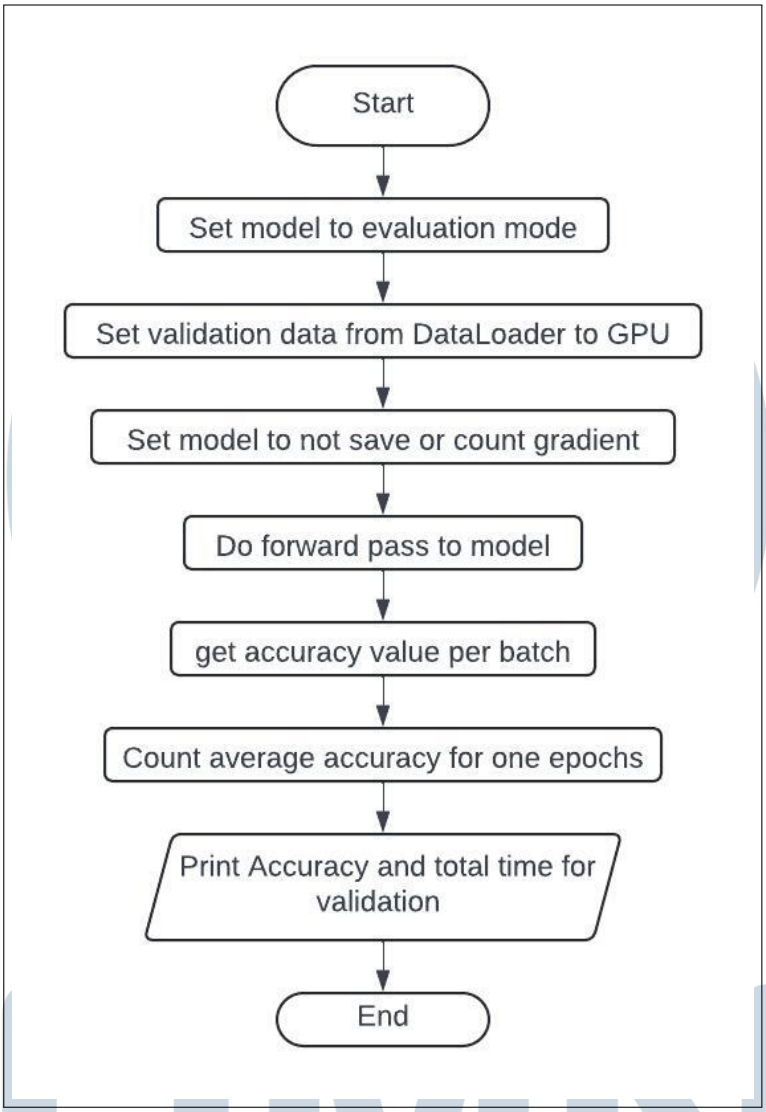


Gambar 3.6. *Train model*



Gambar 3.7. *Fine-tuning process*

Pada Gambar 3.8 validasi model dimulai dengan mengubah model menjadi *evaluation mode* agar model dapat menghasilkan logits berupa hasil prediksi model, setelah itu data untuk validasi yang dipecah oleh DataLoader (sama seperti *data training)* akan dimasukkan ke dalam GPU, dan model diatur agar tidak menghitung gradien. Setelah itu dilakukan *forward pass*, dari proses tersebut didapatkan akurasi per *batch* yang dimasukkan. Nilai akurasi akan diakumulasi untuk seluruh data validasi dan akan dibagi dengan total jumlah data, untuk mendapatkan nilai akurasi dari seluruh data validasi.



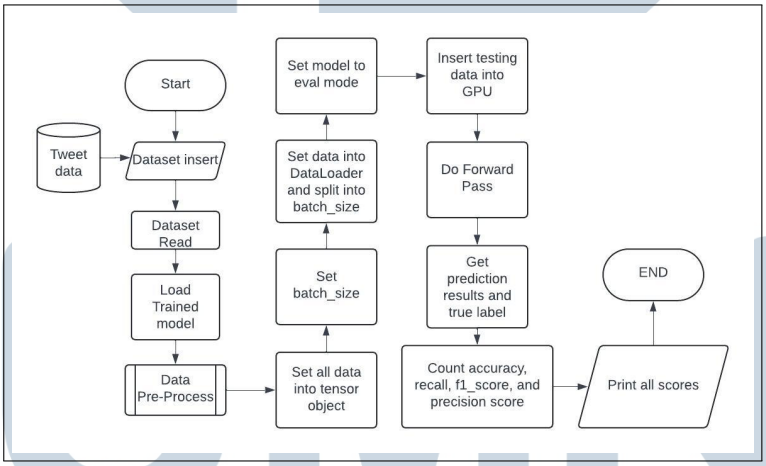
Gambar 3.8. *Validate trained model*

3.4 Model Testing

Pada Gambar 3.9 *testing* dimulai dengan mempersiapkan data penguji yang memiliki parameter data yang sama seperti data pelatihan, yaitu kolom teks dan kolom nilai dari teks. Data akan dimasukkan ke dalam Google Colaboratory, dan akan melewati tahap *pre-process* berupa tokenisasi, *padding* dan *masking* seperti pada Gambar 3.4. Setelah itu, model yang sudah dilatih dipanggil kembali dan dimasukkan ke dalam sebuah fungsi pengecekan *hate speech*. Pada fungsi tersebut, *dataset* yang sudah dimasukkan, dan melewati tahap *pre-process* akan dipecah kembali menggunakan fungsi DataLoader sesuai dengan ukuran *batch*

yang ditentukan. Model akan diatur menjadi mode evaluasi agar dapat melakukan prediksi, lalu data akan dimasukkan ke dalam GPU terlebih dahulu. Setelah data sudah dimasukkan ke dalam GPU, akan dilakukan *forward pass* dengan memasukkan data ke dalam model. Hasil dari *forward pass* tersebut adalah logits yang merupakan hasil prediksi dari model. Hasil logits tiap baris *dataset* akan dikumpulkan untuk menghitung nilai evaluasi dari model. Setelah semua *data testing* berhasil diprediksi, akan dilakukan kalkulasi *accuracy, f1-score, recall* dan *precision* berdasarkan logits dan label yang sebenarnya.

Fungsi yang sudah dibuat tersebut dapat menerima input berupa teks dan label *hate speech* yang benar sebagai data validasi untuk mengukur *accuracy, f1-score, recall* dan *precision*. Setelah *testing* dilakukan maka tingkat nilai evaluasi dari model yang sudah dibuat akan di dapat, dan menjadi hasil dari penelitian ini. Skenario yang akan dilakukan pada *testing* adalah pengujian dengan input teks manual antara model yang sudah dilatih selama 3, 4, 5, 6, 7, dan 8 *epoch*, serta nilai evaluasi dari *data testing*.



Gambar 3.9. *Evaluate model*