BAB III

PELAKSANAAN KERJA MAGANG

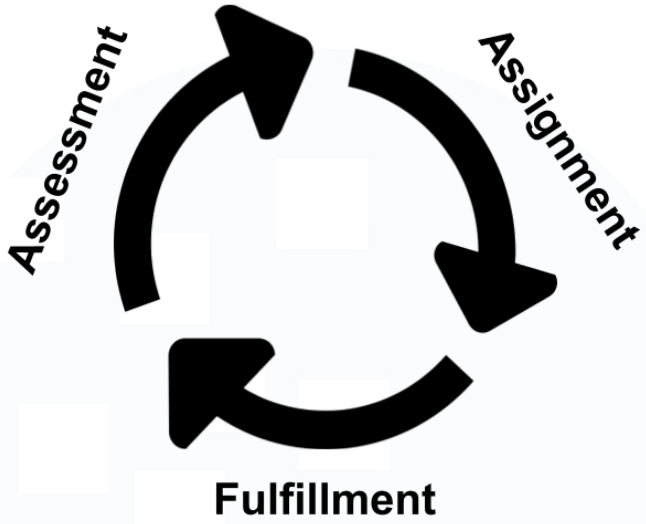
3.1 Kedudukan dan Koordinasi

Divisi MDM pada Kawan Lama Group terbagi atas dua subdivisi *officer* dan *data analyst,* masing-masing dipimpin oleh seorang *manager.* Kedua subdivisi dipimpin oleh seorang *general manager,* yang melapor kepada *Senior VP of Strategic Collaboration Project*. Dalam pelaksanaannya, kerja magang dengan posisi MDM *data analyst intern* disupervisi secara langsung oleh *general manager,* sesuai dengan yang tertera pada portal *human capital* Kawan Lama Group. Alur kerja magang seperti digambarkan pada gambar 3.1. dibawah bersifat iteratif dan terbagi kedalam 3 bagian utama meliputi:

A. ***Assignment*****,** pada tahapan ini penugasan diberikan secara lisan/tulisan oleh *Data Analyst Manager* atau *General Manager* selaku supervisi. Tahapan ini meliputi penyajian informasi terkait tenggat waktu (*deadline*), ketentuan serta penyediaan sumber daya yang dibutuhkan mencakup data yang terkait, instruksi, serta lainnya yang mampu menunjang keberhasilan sebuah proses.

B. ***Fulfillment,*** tahapan selanjutnya adalah proses pemenuhan penugasan. Setiap karyawan di bawah tim MDM diberi kebebasan pada sisi teknik serta alat dalam menyelesaikan tugas dalam kurun waktu yang telah ditentukan.

C. ***Assessment,*** tahapan terakhir dalam alur penugasan adalah proses peninjauan hasil kerja. Proses peninjauan dilakukan secara mingguan melalui *weekly meeting* yang dihadiri oleh *General Manager, Data Analyst Manager,* serta seluruh rekan divisi.



Gambar 3.1. Diagram Alur Kerja Divisi

Dalam kegiatan kerja magang Koordinasi pada divisi dilakukan secara lisan maupun tulisan. Koordinasi secara lisan terjadi pada lingkup kantor, sedangkan secara tulisan dilakukan dengan menggunakan dua *tools*, Whatsapp dan Gmail/Google Chat.

3.2 Tugas dan Uraian Kerja Magang

Sebagai MDM *Data Analyst Intern,* peran utama kami berada pada pembenahan serta peningkatan kualitas data pada skala korporat. Integrasi data produk pada seluruh *business* dibawah naungan Kawan Lama Group mampu meningkatkan daya saing perusahaan melalui pemahaman akan data serta interseksi antar *business unit* pada skala yang lebih besar. Integritas data pada skala korporat dilakukan dengan mengimplementasi *Product Information Management* (PIM). Dalam mendukung kesuksesan implementasi sistem PIM pada Kawan Lama Group, periode magang ini peningkatan kualitas berfokus pada segmentasi produk kedalaman taksonomi yang telah dibuat. Tingginya volume data serta kompleksitas pilihan taksonomi yang mencapai angka 2000, membuat proses segmentasi secara mustahil mustahil untuk diselesaikan pada kurun waktu yang telah ditentukan. Maka dari itu akselerasi proses diperlukan melalui perancangan model *extreme multi-class classifier* (XMC). Adapun penjabaran

pekerjaan berdasarkan dengan periode pengerjaan dapat dilihat pada tabel 3.1. dibawah.

Tabel 3.1. Penjabaran Pekerja per Minggu

No	Pekerjaan	Minggu Ke-	Durasi (Minggu)	Tanggal Mulai	Tanggal Selesai
1	Pengenalan perusahaan, projek, dan tim yang terlibat.	1	1	8 Januari 2024	12 Januari 2024
2	Segmentasi data menyesuaikan dengan taksonomi perusahaan secara manual.	1 - 4	4	8 Januari 2024	2 Februari 2024
3	*Research* pendekatan *machine learning* untuk otomatisasi *extreme multi-class classification* (XMC).	1 - 3	3	8 Januari 2024	26 Januari 2024
4	Pengembangan model *extreme multi-class classification.*	4 - 9	6	2 Februari 2024	8 Maret 2024
5	Pengembangan antarmuka *human-in-the-loop*.	5 - 6	2	5 Februari 2024	16 Februari 2024
6	Segmentasi data artikel dengan pendekatan	6 - 17	12	16 Februari 2024	10 Mei 2024

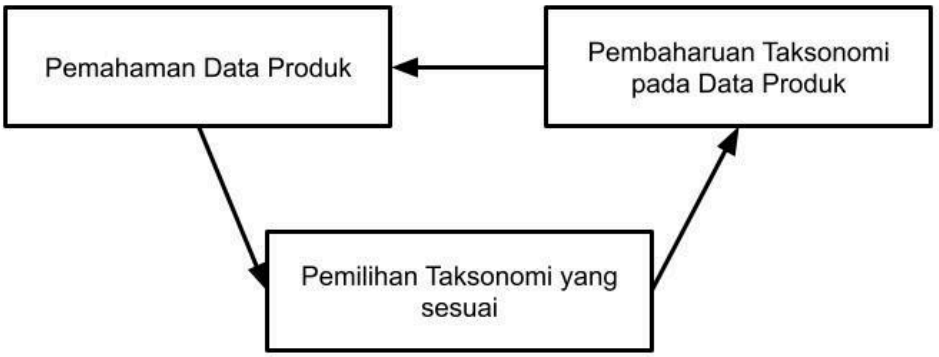
	Human-in-the-loop.				
7	Validasi data pasca segmentasi.	13, 15, 17	3	8, 22 April 2024; 6 Mei 2024	12, 26 April 2024; 10 Mei 2024

3.2.1 Pengenalan Perusahaan dan Projek

Kegiatan pertama sesaat praktik kerja magang dilakukan adalah *Personnel Employee Orientation Program* (PEOP) yang dibawakan oleh divisi *human capital*, kegiatan ini bertujuan untuk memperkenalkan karyawan baru akan lingkungan kerja, aturan, serta tata cara lainnya sebagai panduan selama beraktivitas di lingkungan kerja Kawan Lama Group. Selanjutnya pengenalan akan projek selama periode magang dibawakan oleh *General Manager* selaku *supervisor*, menjabarkan implementasi *Product Information Management* (PIM) serta kontribusi MDM *data analyst intern* kedepannya dalam mendukung keberhasilan implementasi projek. Projek PIM bertujuan untuk mengintegrasi data pada seluruh *business unit* dibawah Kawan Lama Group. Oleh karena itu, pengkategorian atau segmentasi data sesuai dengan taksonomi yang bersifat umum (tidak spesifik hanya pada unit bisnis tertentu) diperlukan.

3.2.2 Segmentasi Data secara Manual

Kegiatan segmentasi data dilakukan secara manual dengan menggunakan Google Spreadsheet sebagai alat bantu. Secara umum, proses segmentasi secara manual dapat dilihat pada gambar 3.2. ini bersifat repetitif dengan tiga tahapan utama:



Gambar 3.2. Diagram Segmentasi Manual

a. Pemahaman data produk

Merupakan tahapan krusial dalam proses analisis bisnis. Tahapan ini dimulai dengan mengidentifikasi jenis produk yang akan disegmentasi. Selanjutnya, proses ini dapat diperdalam dengan melakukan penelusuran mendalam pada internet untuk mengumpulkan informasi terkini mengenai tren dan varian produk dalam pasar yang relevan. Selain itu, membandingkan atribut data produk dengan pilihan taksonomi industri atau standar tertentu membantu dalam menetapkan parameter yang tepat untuk segmentasi. Dengan demikian, pemahaman yang komprehensif terhadap jenis produk dan karakteristiknya sangat penting dalam memastikan analisis data yang akurat dan solusi yang relevan secara bisnis.

b. Pemilihan taksonomi

merupakan langkah yang penting dalam standarisasi representasi produk di seluruh business units Kawan Lama Group. Saat ini, terdapat 2046 pilihan taksonomi yang telah disusun dengan cermat untuk mencakup berbagai kategori produk yang dijual oleh perusahaan. Daftar taksonomi ini merupakan hasil akhir dari upaya

pengembangan yang teliti dan disajikan dalam sebuah Google Spreadsheet yang dapat diakses oleh semua unit bisnis sebagai referensi terbaru. Taksonomi yang lengkap dan terperinci ini memungkinkan setiap produk untuk diidentifikasi dengan jelas berdasarkan karakteristiknya, memudahkan pelaporan, analisis, dan pengelolaan inventaris secara efisien di seluruh jaringan bisnis kami. Dengan memiliki taksonomi yang terstandarisasi, Kawan Lama Group dapat memaksimalkan efektivitas operasional dan memastikan konsistensi dalam pemahaman dan pelaporan data produk di semua level perusahaan.

c. Pembaharuan taksonomi pada data produk

merupakan tahapan terakhir setelah proses pemilihan taksonomi dilakukan. Pada tahap ini, dilakukan penyesuaian atau update taksonomi pada data produk berdasarkan hasil pemilihan taksonomi yang telah ditetapkan sebelumnya. Gambar 3.3. di bawah ini mengilustrasikan struktur data produk yang telah lengkap dengan taksonomi yang diperbarui. Proses ini penting untuk memastikan bahwa data produk dapat dipahami dan dianalisis dengan benar sesuai dengan standar yang telah ditetapkan oleh taksonomi yang terbaru. Dengan demikian, pembaharuan taksonomi pada data produk menjadi langkah kunci dalam memastikan integritas dan akurasi informasi produk di seluruh bisnis unit Kawan Lama Group.

ID	Item Name	Attribute 1	Attribute 2	Attribute 3	Level 1	Level 2	Level 3	Level 4	Leaf Nodes
10581420	Mobile Learning Platform	Platform Type	Content Delivery	User Management	Education	Technology	Access	Remote	LEARN (Learning Enhancement & Resource Network)**
10597234	Smart Irrigation System	Water Source	Automation Features	Monitoring Capabilities	Agriculture	Sustainability	Resource Management	Precision	Aqua Pulse**
10605892	Wearable Health Tracker	Activity Tracking	Health Metrics	Data Integration	Health & Wellness	Technology	Personal Monitoring	Continuous	Vital Link**
10612418	Logistics Management Software	Delivery Management	Inventory Control	Route Optimization	Supply Chain	Operations	Efficiency	Real-time	OptiFlow (Optimized Freight Logistics)
10620175	E-commerce Platform	Product Listing	Payment Processing	Customer Management	Retail	Technology	Sales Channel	Online Storefront	MarketPlace**

Gambar 3.3. Ilustrasi Struktur Data Produk

Proses segmentasi manual ini menghasilkan tingkat produktivitas yang terbatas, dengan rata-rata 830 artikel tersegmentasi per orang per hari. Kegiatan ini berlangsung selama empat minggu, berjalan paralel dengan pengembangan model *Extreme Multi-class Classification* (XMC).

3.2.3 *Research* Metode *Machine Learning*

Terbatasnya tingkat produktivitas yang mampu dicapai melalui proses segmentasi secara manual, mendorong pencarian solusi alternatif berbasis *machine learning* untuk mengakselerasi proses. Tingginya volume data serta kompleksitas kelas klasifikasi (pilihan taksonomi) mengkategorikan jenis klasifikasi ini sebagai *extreme mulit-class classification* (XMC). Adapun beberapa temuan selama proses penelitian dilakukan untuk mengatasi permasalahan XMC meliputi diantaranya:

a. Parabel

Model tersupervisi yang didasarkan pada algoritma Decision Tree yang telah dikembangkan secara khusus untuk menangani kasus di mana terdapat kelas klasifikasi yang tinggi dan tidak terdapat dalam data latih (*unseen label*). Algoritma ini menawarkan keunggulan yang signifikan dalam penanganan kasus-kasus di

mana kelas-kelas baru atau tidak terduga muncul dalam lingkungan pengujian. Misalnya, dalam aplikasi deteksi anomali di lingkungan yang terus berubah, seperti keamanan jaringan atau pemantauan kesehatan, Parabel mampu beradaptasi dengan kehadiran kelas-kelas baru yang belum pernah terlihat sebelumnya. Hal ini membuatnya menjadi pilihan yang sesuai untuk skenario di mana ketersediaan data latih yang representatif sulit dipastikan, namun diperlukan keandalan dan fleksibilitas tingkat tinggi dalam klasifikasi.

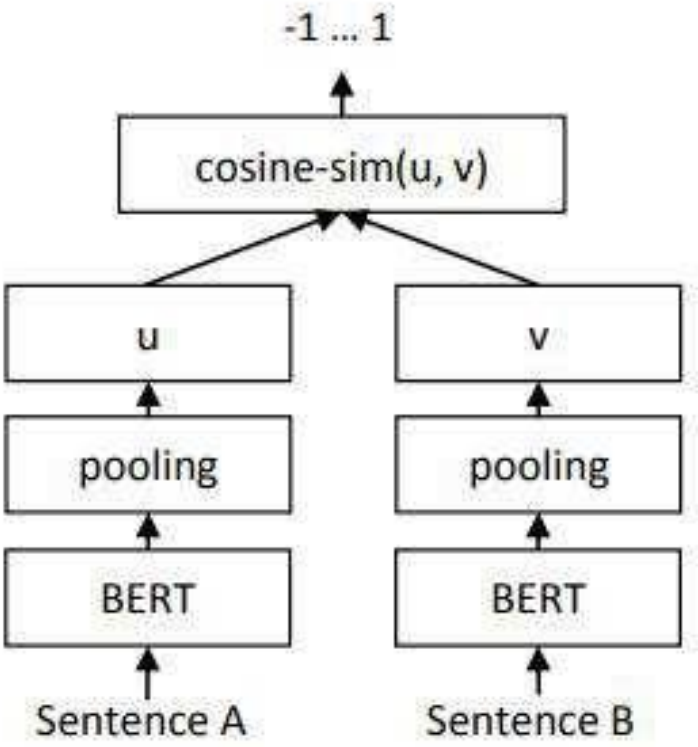
b. ZestXML

Sebuah model klasifikasi berbasis *zero-shot.* Pendekatan model ini mampu menangani permasalahan label yang belum pernah dilihat sebelumnya (*unseen label*) pada data. Metode *zero-shot* memungkinkan model untuk mengenali dan mengklasifikasikan kelas-kelas yang tidak terdapat dalam data pelatihan, dengan menggunakan pengetahuan yang dipelajari secara umum. Namun, implementasi metode ini terkadang terbatas oleh ketersediaan daya komputasi yang memadai. Sebagai contoh, penggunaan ZestXML untuk melatih model klasifikasi *zero-shot* dapat membutuhkan sumber daya komputasi yang besar, sehingga tidak selalu dapat dilakukan dalam skala yang luas. Dengan adanya kendala ini, pendekatan alternatif seperti menggunakan model berbasis transfer learning atau menggunakan representasi kelas yang lebih terbatas mungkin menjadi solusi yang lebih praktis dalam situasi dengan keterbatasan daya komputasi.

c. Siamese BERT (SBERT)

Model yang menerapkan pendekatan berbasis kesamaan untuk mengukur kemiripan semantik (*semantic similarity*) antara dua data dengan memanfaatkan pilihan taksonomi yang tersedia. Model

ini didasarkan pada arsitektur deep learning yang dirancang khusus untuk tugas pengukuran kesamaan semantik. SBERT memanfaatkan Transformer, *neural network* yang telah dipelajari secara mendalam untuk memetakan teks ke dalam representasi vektor yang menyandikan makna dan konteks. Dengan demikian, SBERT dapat secara akurat menentukan seberapa serupa dua data atau dokumen berdasarkan kesamaan makna dan konten mereka, memiliki aplikasi luas dalam berbagai bidang seperti penggalian informasi, analisis teks, dan pengelompokan data. Arsitektur model SBERT dapat dilihat pada gambar 3.4. dibawah.



Gambar 3.4. Arsitektur SBERT Pengukuran Similaritas

Berdasar kepada diagram diatas, untuk mengukur kesamaan dua kalimat yang dibandingkan. Kalimat tersebut akan diproses terpisah oleh SBERT, yang menggunakan model BERT untuk

menangkap hubungan antar kata dan menghasilkan vektor untuk tiap kalimat. Setelah itu, SBERT menghitung kesamaan antar vektor menggunakan metode seperti cosine similarity. Nilai yang dihasilkan (-1 sampai 1) menunjukkan seberapa mirip kedua kalimat, dengan -1 berarti tidak sama sama sekali dan 1 berarti sama persis.

Berdasar kepada tahapan segmentasi yang dilakukan secara manual, serta memperhatikan kecenderungan kesamaan antara atribut data dengan taksonomi yang tersedia menjadi SBERT sebagai pilihan yang sesuai dalam pembangunan model XMC.

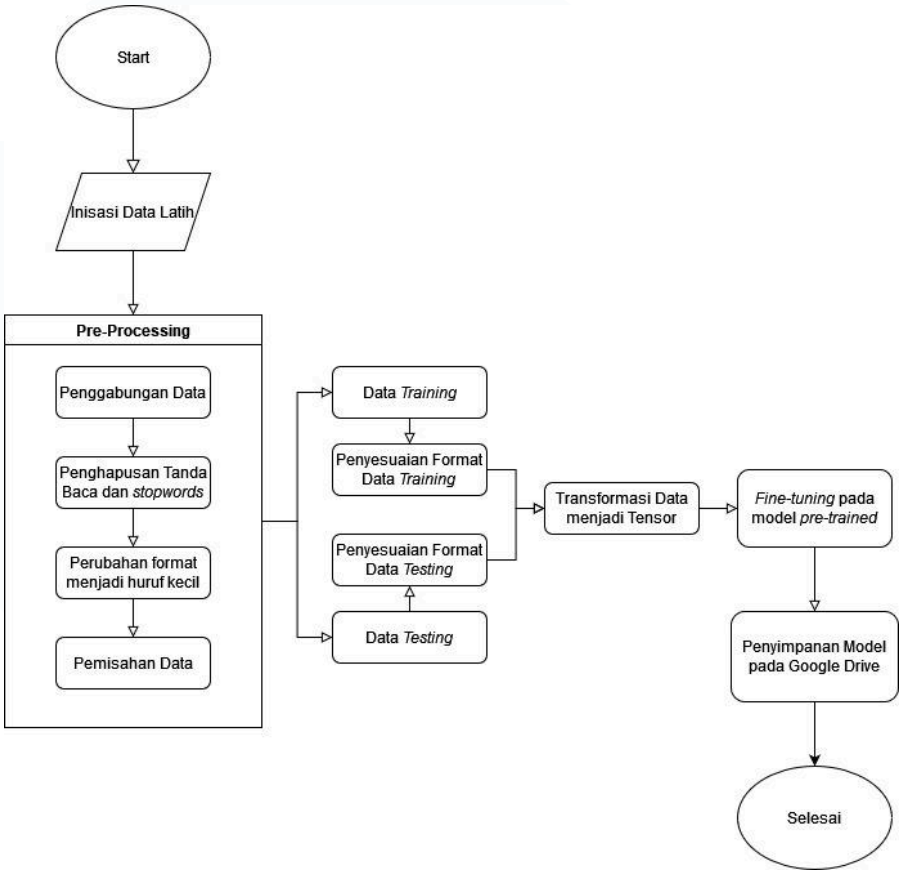
3.2.4 Pengembangan Model XMC

Pengembangan model XMC dengan menggunakan SBERT dilakukan menggunakan Google Colaboratory (Colab) sebagai *intergrated development environment* (IDE), Colab menyediakan layanan *graphical processing unit* (GPU) tidak berbayar yang dibutuhkan untuk menjalankan SBERT sebagai model berbasis *deep learning*. Pengembangan model XMC ini dilakukan secara paralel dengan segmentasi data artikel, berdurasi selama 6 minggu. Pencapaian pengembangan model XMC menggunakan SBERT berdasarkan tahapan pencapaian (*milestones*) dapat dijabarkan sebagai berikut:

a. Pemahaman alur

   Tahap awal periode pengembangan model XMC berfokus pada pemahaman alur proses pelatihan model SBERT (*fine-tuning*) agar sesuai dengan data produk dari Kawan Lama Group yang nantinya akan digunakan dalam proses segmentasi. Proses ini mencakup beberapa langkah penting, antara lain: pemahaman terhadap arsitektur model SBERT dan eksperimen sederhana menggunakan model ini di Colab. Dalam proses ini, kita mempelajari secara mendalam arsitektur dan mekanisme kerja SBERT. Gambar 3.5. menunjukkan *flowchart* fine-tuning pada model SBERT,

memberikan pandangan visual yang jelas terhadap langkah-langkah yang terlibat.



Gambar 3.5. *Flowchart Fine-tuning* Model SBERT

Proses fine-tuning dimulai dengan tahap pre-processing yang bertujuan untuk menyiapkan data artikel agar sesuai dengan kebutuhan model. Langkah pertama adalah menggabungkan artikel yang identik menjadi satu entitas tunggal untuk menghindari duplikasi. Selanjutnya, lima taksonomi yang mencakup level 1, level 2, level 3, level 4, dan leaf nodes digabungkan menjadi satu kategori tunggal untuk mempermudah analisis. Selain itu, tanda baca dihapus dan kata-kata pendek yang tidak bermakna (stopwords) dieliminasi agar fokus pada kata-kata kunci yang relevan. Seluruh huruf dalam teks diubah menjadi huruf kecil

(lowercase) untuk konsistensi dan menghindari perbedaan yang tidak perlu dalam representasi kata. Setelah pre-processing selesai, data dibagi menjadi dua: data latih (train data) dan data uji (test data) rasio 70:30. Pembagian ini penting untuk menguji performa model pada data yang tidak digunakan selama pelatihan (data uji). Proses *fine-tuning* dilakukan dengan memuat model pre-trained menggunakan library Transformer, yang memfasilitasi pengunduhan model dari repositori *cloud* seperti HuggingFace. Model *pre-trained* yang digunakan telah dilatih pada tugas-tugas umum dan kemudian disesuaikan (*fine-tuned*) untuk tugas spesifik yang sedang dihadapi. Selama fine-tuning, data latih dan validasi disesuaikan dengan format yang diperlukan oleh model, termasuk penggunaan loss function yang sesuai. Loss function digunakan untuk mengukur seberapa baik model melakukan prediksi terhadap data validasi selama proses fine-tuning. Setelah proses fine-tuning selesai, parameter dan konfigurasi model yang telah disesuaikan akan disimpan di Google Drive atau lokasi penyimpanan lain yang telah ditentukan sebelumnya. Ini memastikan bahwa model yang telah dilatih dapat diakses dan digunakan kembali untuk tugas yang relevan di masa depan.

b. *Initial testing and result*

Tahapan kedua ini bertujuan untuk menguji performa model menggunakan ⅛ data artikel tahap pertama (50.000 baris data) yang telah melalui proses segmentasi secara manual. *Initial testing* dilakukan dengan menggunakan 3 *loss functions* termasuk diantaranya: Softmax, Multiple Negative Ranking, serta Triplet Loss. Pemilihan ketiga *loss functions* ini berdasar pada karakteristik data serta upaya untuk meningkatkan kemampuan model pada *semantic embedding* (pengukuran kemiripan antara duta data) menyesuaikan dengan data artikel serta taksonomi pada

Kawan Lama. *Fine-tuning* dilakukan dengan menggunakan model *pre-trained* 'ALL-MPNET-BASE-V2', sebagai salah satu model dengan hasil evaluasi terbaik dibandingkan dengan model *pre-trained* lainya. Percobaan dilakukan hanya dengan sekali pengulangan (*epoch*) dengan jumlah sampel data (*batch size)* sebesar 128 data. Hal ini bertujuan untuk mengurangi jumlah waktu yang dilakukan pada proses *fine-tuning,* dengan asumsi peningkatan performa berkorelasi positif dengan peningkatan *epoch* dan pengurangan *batch size*. Hasil *initial testing* dengan mengukur pada data validasi dapat dilihat pada tabel 3.2. dibawah.

Tabel 3.2. Hasil *Initial Testing*

Loss Function	Akurasi@1	Presisi@1
Multiple Negative Ranking	68.5%	68.5%
Softmax	28.9%	25.2%
Triplet Loss	64.3%	30.1%

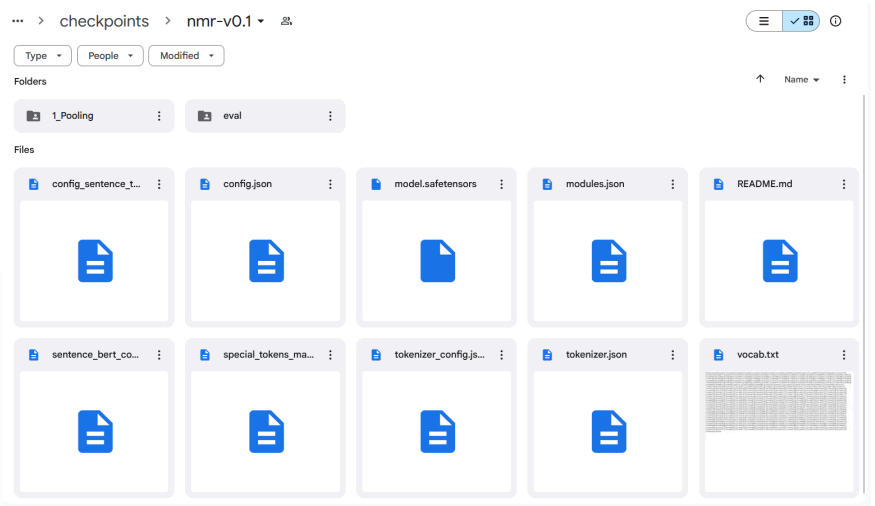
Berdasarkan tabel di atas, implementasi model dengan menggunakan *loss function* Multiple Negative Ranking telah menghasilkan nilai akurasi dan presisi tertinggi, yaitu sebesar 68.5%. Hasil ini lebih baik daripada penggunaan Triplet Loss yang menghasilkan akurasi yang cukup tinggi tetapi presisi yang rendah, serta Softmax yang memiliki akurasi terendah pada 28.9%. Melihat hal ini, diputuskan untuk menggunakan Multiple Negative Ranking sebagai *loss function* dalam pengembangan model, karena kinerjanya yang unggul dalam mencapai akurasi dan presisi yang lebih baik dibandingkan dengan opsi lainnya.

c. *Initial release*

Tahapan ini terbagi menjadi dua bagian utama: meningkatkan kapasitas pelatihan model dan modularisasi program sebelum rilis awal (*initial release*) dilakukan. Peningkatan kapasitas melibatkan *fine-tuning* model dengan menggunakan data dari tahapan sebelumnya, serta meningkatkan jumlah epoch pelatihan dan ukuran *batch* per proses pelatihan untuk meningkatkan akurasi model. Jumlah epoch ditingkatkan hingga 6 kali lipat, dan ukuran batch dikurangi menjadi 32 dari sebelumnya 128 pada tahapan sebelumnya (ukuran batch yang lebih kecil mengakibatkan waktu pelatihan yang lebih lama). Hasil fine-tuning menunjukkan peningkatan setiap epoch, seperti yang terperinci pada Tabel 3.3 di bawah ini, mencapai akurasi sebesar 79,45%. Model ini disimpan di Google Drive dengan nama 'NMR-V0.1'. Struktur *folder* 'NMR-V0.1' yang tersimpan pada Google Drive dapat dilihat pada gambar 3.6. dibawah.

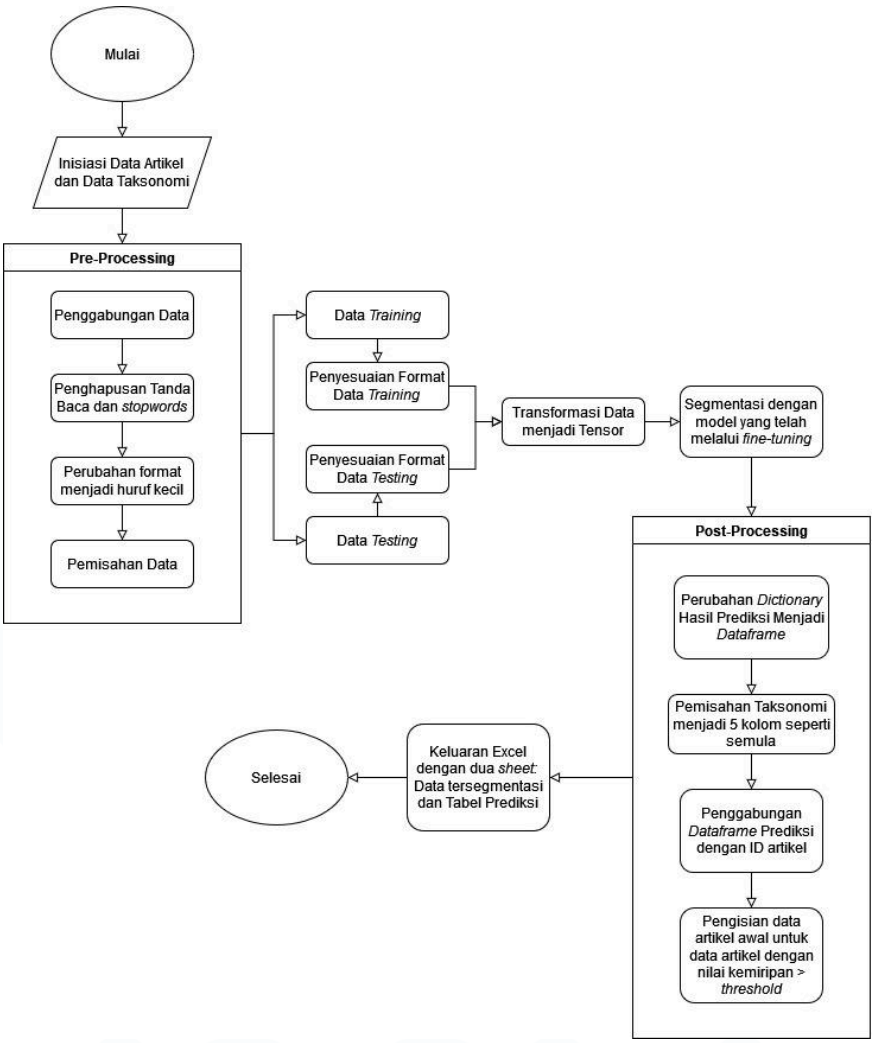
Tabel 3.3. Hasil *Fine-tuning* NMR-V0.1 per *Epoch*

Epoch	**Akurasi@1**	**Presisi@1**
1	66.88%	66.88%
2	73.11%	73.11%
3	76.59%	76.59%
4	78.12%	78.12%
5	78.29%	78.29%
6	79.45%	79.45%



Gambar 3.6. Struktur *Folder* 'NMR-V0.1'

Berdasar kepada struktur *folder* pada gambar diatas, dapat dipahami bahwa parameter model disimpan dalam berbagai *file* konfigurasi dengan ekstensi JSON, meliputi diantaranya konfigurasi untuk Transformers, BERT, *tokenizer*, serta *pooling*. Selanjutnya model SBERT yang telah melalui proses *fine-tuning* disimpan dalam bentuk numerik dengan ekstensi *safetensors*. Format ini memungkinkan penyimpanan serta pemakaian data dalam jumlah besar dengan tetap mempertahankan kecepatan pemrosesan. Selanjutnya dalam rangka mempersiapkan model untuk dapat digunakan dalam proses segmentasi, dilakukan modularisasi fungsi (*function*) ke dalam kelas (*class*). Gambar 3.7. dibawah menampilkan *flowchart* proses segmentasi menggunakan model NMR-V0.1.



Gambar 3.7. *Flowchart* Proses Segmentasi

Berdasar kepada *flowchart* diatas terdapat dua jenis data yang harus disediakan agar proses segmentasi dapat dilakukan: data artikel dan data taksonomi. Kedua data tersebut akan melalui proses *pre-processing* yang serupa dengan tahapan pada proses *fine-tuning*. Data yang telah melalui proses *pre-processing* akan ditransformasi menjadi Tensor, untuk merepresentasikan data teks menjadi numerik. Selanjutnya proses segmentasi dilakukan dengan mengimplementasi fungsi *semantic search* pada *library* Transformers. Fungsi ini bertujuan untuk mencari data taksonomi dengan nilai kemiripan tertinggi dibandingkan dengan data setiap

data artikel. Pada kasus ini secara *default* 5 data dengan nilai kemiripan tertinggi diambil. Fungsi ini mengembalikan dua nilai yakni teks taksonomi dan angka skor kemiripan (dalam rentang 0-100)) dalam bentuk JSON. Penggabungan hasil prediksi dengan data artikel terbagi kedalam tiga tahapan utama yakni: transformasi struktur data JSON menjadi tabel, penggabungan tabel ID artikel dengan tabel prediksi. serta pengisian data artikel (sebelum proses *pre-processing* dilakukan) dengan tabel prediksi pasca penggabungan yang berhasil melewati *threshold* skor kemiripan yang telah ditetapkan. Pertama, transformasi struktur data JSON kedalam tabel dilakukan untuk mempermudah proses selanjutnya yakni penggabungan ID dengan tabel prediksi. Gambar 3.8. dibawah mengilustrasikan tabel hasil prediksi dengan ID artikel.

Article	Category 1	Score 1	Category 2	Score 2	Category 3	Score 3	Category 4	Score 4	Category 5	Score 5
XYZ1001	Electronics & Gadgets	0.92	Home & Living	0.85	Fashion & Accessories	0.78	Outdoor & Adventure	0.75	Health & Wellness	0.7
XYZ1002	Kitchen Essentials	0.88	Kids & Toys	0.82	Fitness & Wellness	0.78	Home Improvement	0.75	Pet Supplies	0.72
XYZ1003	Fashion & Apparel	0.94	Garden & Outdoor	0.9	Office & Stationery	0.85	Beauty & Personal Care	0.8	Arts & Crafts	0.75
XYZ1004	Tech Innovations	0.96	Sports & Fitness	0.92	Music & Entertainment	0.88	Home Appliances	0.85	Gaming & Consoles	0.82
XYZ1005	Home Decor	0.85	Electronics & Accessories	0.8	Fitness & Wellness	0.76	Outdoor & Adventure	0.72	Health & Wellness	0.68
XYZ1006	Fashion & Accessories	0.88	Home & Living	0.84	Kids & Toys	0.78	Pet Supplies	0.75	Arts & Crafts	0.7
XYZ1007	Electronics & Gadgets	0.9	Kitchen Essentials	0.86	Fitness & Wellness	0.82	Outdoor & Adventure	0.78	Health & Wellness	0.74
XYZ1008	Sports & Fitness	0.92	Home Improvement	0.88	Music & Entertainment	0.84	Electronics & Accessories	0.8	Gaming & Consoles	0.76
XYZ1009	Beauty & Personal Care	0.86	Fashion & Apparel	0.82	Arts & Crafts	0.78	Home & Living	0.74	Office & Stationery	0.7

Gambar 3.8. Gambar Tabel Hasil Prediksi dengan ID

Setelah penggabungan dilakukan, maka tahapan terakhir adalah pengisian tabel data artikel awal dengan taksonomi pertama pada hasil prediksi yang berhasil melewati *threshold* (secara *default* 70%). Namun, sebelum diisi ke dalam data artikel awal, seperti yang terlihat pada Gambar 3.6 diatas, taksonomi yang dihasilkan masih tergabung ke dalam satu teks. Maka dari itu, pemisahan taksonomi menjadi semula dengan total 5 kolom (Level 1, Level 2, Level 3, Level 4, dan *Leaf Nodes*) perlu dilakukan. Tahapan terakhir adalah proses ekspor ke dalam Excel yang berisikan dua *sheet, sheet pertama* berisikan data yang telah tersegmentasi dan

kedua berisikan hasil prediksi dan skornya. Berakhirnya proses *fine-tuning* model serta perancangan modularisasi program dilanjutkan dengan *initial release*, proses ini disertai pemberian oleh MDM *General Manager* dan MDM *Data Analyst Manager* dalam penggunaan serta pengujian model dalam proses segmentasi secara parsial dengan bantuan manusia. Program yang telah termodularisasi terbagi kedalam tiga *class*, LAMHelper, LAM, serta SBERT_MNR. *Class* LAMHelper merupakan *class* penunjang lainnya yang berisikan fungsi-fungsi. Tujuan pembuatan *class* ini adalah mengurangi adanya repetisi dalam program. Adapun penjabaran fungsi tersebut adalah sebagai berikut:

- *cols_exist*(), fungsi ini bertujuan memeriksa apakah kolom atau daftar kolom ada di dalam *DataFrame* tertentu. Fungsi ini memvalidasi tipe input dari kolom, memastikan nya berupa *string* (mewakili nama kolom tunggal) atau daftar string (mewakili beberapa nama kolom). Jika validasi gagal, fungsi ini akan menimbulkan *TypeError* (Kesalahan Tipe Data). Implementasi fungsi ini dapat dilihat pada gambar 3.9. dibawah.

```
#=== GENERAL
def cols_exist(self, data, cols):
  if isinstance(cols, str):
      return cols in data.columns
  elif isinstance(cols, list):
      return all(col in data.columns for col in cols)
  else:
      raise TypeError("Cols must be either a string or a list of strings")
```

Gambar 3.9. Fungsi *cols_exist*() pada LAMHelper

- get_rows_missing_values() berfungsi mengidentifikasi baris dalam *DataFrame* (data) yang tidak ada di *DataFrame* lain (target_data). Fungsi ini beriterasi melalui setiap baris dalam data dan memeriksa apakah baris tersebut ada di *target_data*. Jika baris tidak ditemukan, indeksnya ditambahkan ke dalam daftar, yang kemudian

dikembalikan. Selanjutnya fungsi *to_string* membantu dalam mengonversi *DataFrame* (data) menjadi representasi string. Fungsi ini mengisi nilai yang hilang dengan string kosong ('') dan kemudian mengonversi setiap kolom menjadi tipe string menggunakan *.astype('str')*. Implementasi fungsi ini dapat dilihat pada gambar 3.10 dibawah.

```
def get_rows_missing_values(self, data, target_data):
  return [i for i, row in enumerate(data) if row not in target_data]
```

Gambar 3.10. Fungsi *get_rows_missing_values*() pada LAMHelper

- *join_no_duplicate*() menggabungkan teks dari baris (*row*) menjadi satu string tunggal sambil menghilangkan duplikat dan mempertahankan urutannya. Fungsi ini menggunakan ekspresi regular (re.*split*) untuk memecah teks berdasarkan spasi putih, karakter khusus (/, pipa, dan karakter non-alfanumerik), menciptakan daftar kata. Kemudian, fungsi ini beriterasi melalui kata-kata tersebut, mengubahnya menjadi huruf kecil, menghilangkan spasi, dan menambahkannya ke daftar (unique_words) hanya jika kata tersebut belum pernah ditemui sebelumnya. Terakhir, fungsi ini menggabungkan kembali kata-kata unik tersebut dengan spasi dan mengembalikan string gabungan. Implementasi fungsi ini dapat dilihat pada gambar 3.11 dibawah.

```
def join_no_duplicate(self, row):
  words = list()
  unique_words = []

  for text in row:
    words.extend(re.split(r'\s|/|[^\w\s]', text))

  for word in words:
    word = word.lower().strip()
    if word not in unique_words:
      unique_words.append(word)

  return ' '.join(unique_words).strip()
```

Gambar 3.11. Fungsi *join_no_duplicate*() pada LAMHelper

- *get_unique_row*(), Fungsi ini mengambil baris unik dari *DataFrame* (data), memastikan tidak ada duplikat. Pertama, fungsi ini memanggil fungsi *get_key_col*() untuk menentukan kolom kunci yang digunakan untuk identifikasi. Kemudian, fungsi ini menggunakan metode *.drop_duplicates*() dengan parameter subset yang disetel ke kolom kunci untuk menghapus duplikat. Terakhir, fungsi ini mengatur ulang indeks untuk membuat *DataFrame* baru dengan baris unik. Implementasi fungsi ini dapat dilihat pada gambar 3.12 dibawah.

```
def get_unique_row(self, data):
  key_col = self.get_key_col(data)
  return data.drop_duplicates(subset=key_col).reset_index()
```

Gambar 3.12. Fungsi *get_unique_row*() pada LAMHelper

- *split_data*(), Fungsi ini membagi *DataFrame* (data) menjadi set pelatihan, validasi, dan pengujian untuk tujuan pembelajaran mesin. Fungsi ini menggunakan fungsi *train_test_split* dari library Scikit-learn (Sklearn.*model_selection*). Di sini, fungsi ini menentukan rasio set pengujian (*test_ratio*) dan membagi data sesuai dengan itu. Fungsi ini juga menggunakan random_state (*random_state*=42) untuk memastikan reprodusibilitas saat

membagi data. Fungsi ini mengembalikan *DataFrame* pelatihan, validasi, dan pengujian. Implementasi fungsi ini dapat dilihat pada gambar 3.13 dibawah.

```
def split_data(self, data, test_ratio):
  data_train, data_temp = train_test_split(data, test_size=test_ratio,
                                           random_state=42)

  val_ratio = 0.5 * (1 - test_ratio)
  data_val, data_test = train_test_split(data_temp, test_size=val_ratio,
                                         random_state=42)

  return data_train, data_val, data_test
```

Gambar 3.13. Fungsi *split_data*() pada LAMHelper

- *merge_true_predict*(), Fungsi ini menggabungkan dua *DataFrame* (*data_true* dan *data_predict*), biasanya berisi label sebenarnya dan label prediksi dari tugas klasifikasi. Fungsi ini mengasumsikan adanya kolom taksonomi bernama *taxonomy_label* di *DataFrame data_true*. Fungsi ini membuat kolom baru bernama {*taxonomy_label*}*_predict* di *data_predict* untuk menyimpan label prediksi. Kemudian, fungsi ini menggabungkan kedua *DataFrame* sepanjang sumbu 1 (kolom) untuk membuat *DataFrame* gabungan yang berisi label sebenarnya dan prediksi. Implementasi fungsi ini dapat dilihat pada gambar 3.14 dibawah.

```
def merge_true_predict(self, data_true, data_predict, taxonomy_label):
  LABEL = taxonomy_label
  LABEL_PREDICT = LABEL + '_predict'

  data_true = self.get_taxonomy_merged(data_true, LABEL)
  data_predict = self.get_taxonomy_merged(data_predict, LABEL_PREDICT)

  data_merged = pd.concat([data_true, data_predict[LABEL_PREDICT]], axis=1)
  return data_merged
```

Gambar 3.14. Fungsi *merge_true_predict*() pada LAMHelper

- *get_taxonomy_cols*(), Fungsi ini mengembalikan daftar yang mewakili nama kolom taksonomi standar, termasuk 'Level 1', 'Level 2', 'Level 3', 'Level 4', dan '*Leaf Nodes*'.

Implementasi fungsi ini dapat dilihat pada gambar 3.15 dibawah.

```
def get_taxonomy_cols(self):
  return ['Level 1','Level 2','Level 3', 'Level 4', 'Leaf Nodes']
```

Gambar 3.15. Fungsi *get_taxonomy_cols*() pada LAMHelper

- *get_taxonomy_merged*(), Fungsi ini memastikan DataFrame (data) memiliki kolom yang berisi level taksonomi gabungan. Jika kolom bernama *taxonomy_label* sudah ada, fungsi ini mengembalikan data apa adanya. Jika tidak, fungsi ini mengasumsikan adanya kolom Level terpisah (lihat *get_taxonomy_cols*) dan menggabungkannya menjadi satu kolom bernama *taxonomy_label* menggunakan fungsi lambda dan *.apply*. Fungsi lambda menggabungkan nilai dari setiap kolom Level dengan spasi dan menghilangkan spasi awal/akhir. Implementasi fungsi ini dapat dilihat pada gambar 3.16 dibawah.

```
def get_taxonomy_merged(self, data, taxonomy_label):
  if self.cols_exist(data, taxonomy_label):
    return data

  taxonomy_subset = self.get_taxonomy_cols()[:-1]
  data[taxonomy_label] = data[taxonomy_subset].apply(
      lambda x: ' '.join(x).strip(), axis=1).reset_index(drop=True)
  return data
```

Gambar 3.16. Fungsi *get_taxonomy_merged*() pada LAMHelper

- *get_negative_pair*(), Fungsi ini menghasilkan kolom pasangan negatif di dalam *DataFrame* (*data_true*). Fungsi ini mengasumsikan adanya kolom *'Taxonomy* 1' dan kolom *taxonomy_label* di *data_true*, bersama dengan *DataFrame* (*predictions*) yang berisi label prediksi. Fungsi ini membuat DataFrame sementara dengan menggabungkan *data_true*

dengan representasi. Implementasi fungsi ini dapat dilihat pada gambar 3.17 dibawah.

```
def get_negative_pair(self, data_true, predictions, taxonomy_label):
  suffix = '_predict'

  data_true_merged = pd.concat([data_true, self.to_string(predictions)],
                               axis=1)

  negative_pair_list = [row['Taxonomy 1'] if row['Taxonomy 1'] !=
                        row[taxonomy_label] else row['Taxonomy 2'] for index,
                        row in data_true_merged.iterrows()]

  #List of negative pair into column
  data_true[taxonomy_label+suffix] =  negative_pair_list
  print(data_true.head(10))
  return data_true
```

Gambar 3.17. Fungsi *get_negative_pair*() pada LAMHelper

- *get_key_col*(), fungsi ini bertujuan untuk mengidentifikasi kolom kunci dalam kumpulan data tertentu. Ini menetapkan kunci default bernama 'Artikel'. Jika dataset memiliki kolom bernama eksplisit 'Artikel', fungsi ini mengembalikan 'Artikel' sebagai kolom kunci. Jika tidak, fungsi ini mengambil kolom pertama yang ada dalam dataset dan menetapkannya sebagai kolom kunci. Implementasi fungsi ini dapat dilihat pada gambar 3.18 dibawah.

```
def get_key_col(self, data):
  KEY = 'Article'
  #Default article, first index if none Article column
  return KEY if KEY in data else data.columns[0]
```

Gambar 3.18. Fungsi *get_key_col*() pada LAMHelper

- *get_subset_cols*(), Fungsi ini bertanggung jawab untuk menghasilkan sub kumpulan kolom dari kumpulan data yang disediakan. Ini dilakukan dengan iterasi melalui semua kolom dan membuat daftar baru yang mengecualikan kolom yang diidentifikasi melalui fungsi *get_taxonomy_cols* (mungkin didefinisikan di tempat lain di kelas). Daftar ini mewakili kolom inti dari kumpulan

data. Secara opsional, dengan menyetel argumen *non_key* ke *True*, fungsi ini semakin mempersempit sub kumpulan kolom. Dalam kasus ini, fungsi ini mengambil kolom kunci menggunakan fungsi *get_key_col* dan menghilangkan kolom tertentu tersebut dari subkumpulan yang dihasilkan sebelumnya, memastikan kolom kunci tidak disertakan dalam subkumpulan akhir. Implementasi fungsi ini dapat dilihat pada gambar 3.19 dibawah.

```
def get_subset_cols(self, data, non_key=False):
  subset_cols = [col for col in data.columns if
                 col not in self.get_taxonomy_cols()]

  if non_key is True:
    key_col = self.get_key_col(data)
    subset_cols = [col for col in subset_cols if col != key_col]

  return subset_cols
```

Gambar 3.19. Fungsi *get_subset_cols*() pada LAMHelper

- *get_article_merged*(), fungsi ini memeriksa keberadaan kolom *article_label*. Kolom ini menandakan bahwa data sudah digabung sebelumnya. Jika tidak ada, fungsi ini mengelompokkan data berdasarkan kolom non-taksonomi. Di dalam setiap kelompok, fungsi ini menggabungkan nilai-nilai dari kolom non-kunci dan non-taksonomi (menggunakan fungsi *join_no_duplicate* yang diasumsikan untuk menghindari duplikat). Hasil penggabungan tersebut disimpan di kolom baru bernama *article_label*. Terakhir, data asli digabungkan dengan data yang sudah diproses berdasarkan kolom kunci, menghilangkan duplikat dan berpotensi menangani penggabungan taksonomi melalui fungsi lain (*get_taxonomy_merged*) jika berlaku. Implementasi fungsi ini dapat dilihat pada gambar 3.20 dibawah.

```
def get_article_merged(self, data, article_label, taxonomy_label):
  if self.cols_exist(data, article_label):
    return data

  key_col = self.get_key_col(data)
  taxonomy_cols = self.get_taxonomy_cols()
  subset_cols = self.get_subset_cols(data)
  non_key_subset_cols = self.get_subset_cols(data, non_key=True)

  data_group_key = data[subset_cols].groupby(key_col, group_keys=False)
  data_group_key = data_group_key.apply(
      lambda x: x.apply(
          lambda row: self.join_no_duplicate(row) if
          row.name != key_col else row, axis=0)).reset_index(drop=True)
  data_group_key[article_label] = data_group_key[non_key_subset_cols].apply(
      self.join_no_duplicate, axis=1).reset_index(drop=True)

  data_merged = pd.merge(data, data_group_key[[key_col, article_label]],
                         on=key_col, how='left')
  data_merged = data_merged.drop_duplicates(
      subset=key_col).reset_index(drop=True)

  if self.cols_exist(data_merged, taxonomy_cols):
    data_merged = self.get_taxonomy_merged(data_merged, taxonomy_label)

  return data_merged
```

Gambar 3.20. Fungsi *get_article_merged*() pada LAMHelper

Selanjutnya *class* SBERT_MNR bertujuan sebagai *class* untuk inferensi model berbasis SBERT baik *pre-trained* ataupun yang telah melalui proses *fine-tuning*. Adapun penjabaran daripada fungsi SBERT_MNR adalah sebagai berikut:

- *__init__*(), fungsi ini berperan sebagai konstruktor untuk kelas yang diinisialisasi dengan parameter *model_name*, *article*, *taxonomy*, *article_label*, dan *taxonomy_label*. Saat objek kelas dibuat, konstruktor akan menetapkan nilai-nilai ini ke dalam atribut-atribut objek. Pertama, *model_name*, *article_label*, dan *taxonomy_label* disimpan sebagai atribut objek. Selanjutnya, objek LAMHelper dibuat untuk memberikan utilitas pemrosesan data. article dan taxonomy dari *input* diterjemahkan menjadi *string* menggunakan metode *to_string* dari LAMHelper, dan hasilnya disimpan sebagai atribut article dan taxonomy. Kemudian, dilakukan

penggabungan artikel (*article_merged*) dan taksonomi (*taxonomy_merged*) menggunakan metode *get_article_merged*() dan *get_taxonomy_merged*() dari LAMHelper. Selain itu, data artikel dipisahkan menjadi data pelatihan, validasi, dan uji menggunakan metode *split_data*() dari LAMHelper dengan rasio uji sebesar 0,15. Terakhir, atribut device ditentukan berdasarkan ketersediaan CUDA untuk perangkat akselerasi GPU pada sistem dengan menguji torch.cuda.*is_available*(). Jika CUDA tersedia, device diatur ke "cuda"; jika tidak, diatur ke "cpu". Dengan demikian, konstruktor ini melakukan berbagai inisialisasi dan pengaturan awal objek kelas dengan memanfaatkan utilitas yang disediakan oleh LAMHelper. Implementasi fungsi ini dapat dilihat pada gambar 3.21. dibawah.

```
def __init__(self, model_name, article, taxonomy, article_label, taxonomy_label):
    self.model_name = model_name
    self.article_label = article_label
    self.taxonomy_label = taxonomy_label

    self.helper = LAMHelper()
    self.article = self.helper.to_string(article)
    self.taxonomy = self.helper.to_string(taxonomy)
    self.article_merged = self.helper.get_article_merged(self.article, self.article_label, self.taxonomy_label)
    self.taxonomy_merged = self.helper.get_taxonomy_merged(self.taxonomy, self.taxonomy_label)
    self.article_train, self.article_val, self.article_test = self.helper.split_data(self.get_article_merged(), test_ratio=0.15)

    self.device = "cuda" if torch.cuda.is_available() else "cpu"
```

Gambar 3.21. Fungsi *__init__*() pada SBERT_MNR

- *get_*, seluruh fungsi enkapsulasi dirancang untuk mengakses berbagai atribut dari objek kelas ini. Metode *get_artic*() digunakan untuk mengembalikan nilai dari atribut article. Metode *get_taxonomy*() mengembalikan nilai dari atribut taxonomy. Kemudian, *get_taxonomy_label*() mengembalikan nilai atribut *taxonomy_label*. Metode *get_article_merged*() mengembalikan nilai atribut *article_merged*, sedangkan *get_taxonomy_merged* mengembalikan nilai atribut

taxonomy_merged. Terakhir, metode *get_taxonomy_merged_list*() adalah metode yang mengembalikan daftar (*list*) dari nilai atribut *taxonomy* berdasarkan kunci yang terdapat dalam atribut *taxonomy_label*. Metode-metode ini dirancang untuk memastikan bahwa akses ke atribut-atribut ini dilakukan dengan cara yang terencana dan terstruktur, serta mempertahankan prinsip enkapsulasi dalam pemrograman berorientasi objek. Implementasi fungsi ini dapat dilihat pada gambar 3.22 dibawah.

```
#=== Encapsulation Method
def get_article(self):
  return self.article

def get_taxonomy(self):
  return self.taxonomy

def get_taxonomy_label(self):
  return self.taxonomy_label

def get_article_merged(self):
  return self.article_merged

def get_taxonomy_merged(self):
  return self.taxonomy_merged

def get_taxonomy_merged_list(self):
  return self.get_taxonomy()[self.get_taxonomy_label()]
```

Gambar 3.22. Fungsi enkapsulasi pada SBERT_MNR

- *get_negative_pair_col*, fungsi ini memeriksa apakah ada kolom tertentu dalam data artikel yang digabungkan yang menunjukkan pasangan negatif untuk pelatihan. Ini membangun nama kolom dengan menambahkan sufiks ke label taksonomi. Mengembalikan nama kolom jika ada, jika tidak kosong (*None*). Implementasi fungsi ini dapat dilihat pada gambar 3.23. dibawah.

```
def get_negative_pair_col(self):
  SUFFIX = '_predict'
  negative_pair_col = self.get_taxonomy_label() + SUFFIX

  if self.helper.cols_exist(self.get_article_merged(), negative_pair_col):
    return negative_pair_col

  else:
    return None
```

Gambar 3.23. Fungsi *get_negative_pair_col*() pada SBERT_MNR

- *_generate_val_data*(), fungsi ini menyiapkan data untuk validasi. Ini membuat *dictionary* untuk *query* IR (*Information Retrieval*) (label artikel), corpus (entri taksonomi), dan dokumen relevan. Ini iterasi melalui data validasi, membuat entri dalam *dictionary query* dengan label artikel sebagai kunci. Untuk setiap artikel, ia menemukan label taksonomi yang sesuai dan menambahkan ID dokumen relevan (indeks dalam *corpus*) ke *dictionary* dokumen relevan. Implementasi fungsi ini dapat dilihat pada gambar 3.24. dibawah.

```
def _generate_val_data(self):
  ir_queries = {}
  ir_corpus = { index: value for index, value in enumerate(self.get_taxonomy_merged_list())}
  ir_relevant_docs = {}

  for index, row in self.article_val.iterrows():
    ir_queries[index] = row[self.article_label]
    ir_relevant_docs[index] = set()

    for key, value in ir_corpus.items():
      if row[self.taxonomy_label] == value:
        ir_relevant_docs[index].add(key)

  return ir_queries, ir_corpus, ir_relevant_docs
```

Gambar 3.24. Fungsi *_generate_val_data*() pada SBERT_MNR

- *_generate_train_data*(), fungsi ini menghasilkan data pelatihan sebagai *InputExamples* sekaligus memeriksa keberadaan kolom pasangan negatif. Jika kolom negatif ada, pembuatan sampel pelatihan dengan tiga teks: label

artikel, label taksonomi, dan label pasangan negatif dari kolom yang diidentifikasi. Jika kolom pasangan negatif tidak tersedia, ini membuat sampel pelatihan hanya dengan label artikel dan taksonomi. Implementasi fungsi ini dapat dilihat pada gambar 3.25. dibawah.

```
def _generate_train_data(self):
  train_samples = []

  negative_pair_col = self.get_negative_pair_col()

  for index, row in self.article_train.iterrows():
    if negative_pair_col is None:
      train_samples.append(InputExample(texts=[ row[self.article_label],
                                                row[self.taxonomy_label] ]))
    else:
      train_samples.append(InputExample(texts=[ row[self.article_label],
                                                row[self.taxonomy_label],
                                                 row[negative_pair_col] ]))

  return train_samples
```

Gambar 3.25. Fungsi *_generate_train_data*() pada SBERT_MNR

- *fine_tune*(), fungsi ini melakukan pelatihan pada model inti dengan mengonfigurasi model SentenceTransformer berdasarkan nama model yang ditentukan serta menetapkan panjang urutan maksimum. Proses ini melibatkan pembuatan lapisan pooling yang digabungkan dengan model embedding kata dari SentenceTransformer. Selanjutnya, fungsi menghasilkan data pelatihan menggunakan metode *_generate_train_data* dan menggunakan *DataLoader* untuk pemrosesan *batch* yang efisien, serta menentukan fungsi MultipleNegativesRankingLoss untuk tujuan pelatihan. Data validasi juga disiapkan dengan metode *_generate_val_data* dan menggunakan objek InformationRetrievalEvaluator untuk evaluasi hasil. *Hyperparameter* seperti langkah pemanasan untuk

optimizer ditetapkan, kemudian fungsi fine_tune() memanggil metode *fit* dari model SentenceTransformer, yang melibatkan proses pelatihan dengan menyediakan data pelatihan, fungsi *loss*, *evaluator*, jumlah *epochs*, langkah pemanasan, serta opsi lain seperti penyimpanan model terbaik dan penggunaan *Automatic Mixed Precision* (AMP) untuk pelatihan yang lebih cepat. Implementasi fungsi ini dapat dilihat pada gambar 3.26. dibawah.

```
def fine_tune(self, output_path, train_batch_size, max_seq_length, num_epochs):
  word_embedding_model = models.Transformer(self.model_name,
                                            max_seq_length=max_seq_length)
  pooling_model = models.Pooling(
      word_embedding_model.get_word_embedding_dimension(),
      pooling_mode='mean')
  model = SentenceTransformer(modules=[word_embedding_model,
                                       pooling_model], device=self.device)

  #=== Generate train data and load in DataLoader
  train_data = self._generate_train_data()
  train_dataloader = DataLoader(train_data, shuffle=True,
                                batch_size=train_batch_size)
  train_loss = losses.MultipleNegativesRankingLoss(model)

  #=== Generate val data and evaluator
  ir_queries, ir_corpus, ir_relevant_docs = self._generate_val_data()
  val_evaluator = InformationRetrievalEvaluator(
      ir_queries,
      ir_corpus,
      ir_relevant_docs,
      score_functions={"cos_sim": util.cos_sim}, map_at_k=[1])

  warmup_steps = math.ceil(len(train_dataloader) * num_epochs * 0.1)
  #=== Fit Model
  model.fit(
      train_objectives=[(train_dataloader, train_loss)],
      evaluator = val_evaluator,
      epochs = num_epochs,
      warmup_steps = warmup_steps,
      save_best_model = True,
      output_path = output_path,
      use_amp = True
  )
```

Gambar 3.26. Fungsi *fine_tune*() pada SBERT_MNR

- *inference*(), fungsi ini melakukan inferensi model pada data yang tidak terlihat. Pertama-tama, fungsi ini mencantumkan model yang digunakan, yaitu model SentenceTransformer yang sudah terlatih sebelumnya. Model ini dimuat untuk mengkodekan baik *query* (label artikel) maupun korpus

(entri taksonomi) dari dataset yang digabungkan. Dengan mengkodekan input tersebut menggunakan model, fungsi ini melakukan pencarian semantik menggunakan vektor yang terenkoding untuk mengidentifikasi entri paling mirip dalam korpus untuk suatu query. Selanjutnya, fungsi ini melintasi hasil pencarian dan menyusun daftar prediksi yang berisi label taksonomi beserta skornya. Jumlah prediksi yang dikembalikan dibatasi berdasarkan argumen *max_preds*, sehingga mengoptimalkan efisiensi keluaran. Implementasi fungsi ini dapat dilihat pada gambar 3.27. dibawah.

```
def inference(self, max_preds):
  #=== Model Initialization & Data Embedding
  print("Running Inference on: ", self.model_name)
  model = SentenceTransformer(self.model_name, device=self.device)
  query = self.get_article_merged()[self.article_label]
  corpus = self.get_taxonomy_merged()[self.taxonomy_label]

  query_embedding = model.encode(query, convert_to_tensor=True)
  corpus_embedding = model.encode(corpus, convert_to_tensor=True)

  #=== Semantic Search
  preds_semantic = util.semantic_search(query_embedding, corpus_embedding)

  predictions_arr = []
  for index, pred in enumerate(preds_semantic):
    temp_arr = []

    count = 0
    for item in pred:
      id, score = item['corpus_id'], item['score']
      taxonomy = corpus[id]
      temp_arr.append({"taxonomy": taxonomy, "score": score})
      count += 1

      if count == max_preds:
        break;

    # print(f"Prediction at {index}:", temp_arr)
    predictions_arr.append(temp_arr)

  return predictions_arr
```

Gambar 3.27. Fungsi *inference*() pada SBERT_MNR

Selanjutnya *class* LAM sebagai *class* untuk inferensi model berbasis SBERT baik *pre-trained* ataupun yang telah melalui

proses *fine-tuning*. Adapun penjabaran daripada fungsi SBERT_MNR adalah sebagai berikut:

- *__init__*(), fungsi ini adalah fungsi inisialisasi yang dipanggil ketika objek kelas LAM dibuat. Ia mewarisi fungsi inisialisasi dari kelas induk SBERT_MNR dan kemudian mendefinisikan atribut predictions dengan nilai awal *None*. Implementasi fungsi ini dapat dilihat pada gambar 3.28. dibawah.

```
def __init__(self, model_name, article, taxonomy, article_label, taxonomy_label):
  super().__init__(model_name, article, taxonomy, article_label, taxonomy_label)
  self.predictions = None
```

Gambar 3.28. Fungsi *__init__*() pada LAM

- *_arr_to_df*(), fungsi ini berfungsi untuk mengubah array prediksi menjadi sebuah *DataFrame*. Ia menerima dua argumen: *predictions_arr* yang berisi *array* prediksi dan *max_preds* yang menentukan jumlah prediksi maksimum yang akan ditampilkan. Fungsi ini mengekstrak taksonomi dan skor dari array prediksi, kemudian memastikan setiap *sub-array* memiliki panjang yang sama dengan max_preds dengan menambahkan elemen kosong jika diperlukan. Terakhir, fungsi ini membuat *DataFrame* dengan kolom yang berisi taksonomi dan skor prediksi. Implementasi fungsi ini dapat dilihat pada gambar 3.29 dibawah.

```python
def _arr_to_df(self, predictions_arr, max_preds):
  # Extract taxonomies and scores from predictions_arr
  taxonomies = [[obj.get('taxonomy', '') for obj in arr] for arr in predictions_arr]
  scores = [[obj.get('score', np.nan) for obj in arr] for arr in predictions_arr]

  # Pad the arrays to ensure each sublist has length equal to max_preds
  taxonomies = [cats + [''] * (max_preds - len(cats)) for cats in taxonomies]
  scores = [scrs + [np.nan] * (max_preds - len(scrs)) for scrs in scores]

  # Create dictionary for DataFrame
  df_dict = {f'Taxonomy {i+1}': [cat[i] for cat in taxonomies] for i in range(max_preds)}
  df_dict.update({f'Score {i+1}': [scr[i] for scr in scores] for i in range(max_preds)})

  # Create DataFrame
  df_preds = pd.DataFrame(df_dict)

  # article_key_col = self.helper.get_key_col(self.get_article())
  # df_preds = pd.concat([self.get_article_merged(), df_preds], axis=1)
  return df_preds
```

Gambar 3.29. Fungsi *_arr_to_df_*() pada LAM

- *_fill_wrong_taxonomy*(), fungsi ini berfungsi untuk menangani kasus ketika data berisi nilai kosong pada kolom taksonomi. Ia menerima argumen data yang berupa *DataFrame* dan mengembalikan *DataFrame* yang sudah terisi dengan nilai kosong pada kolom taksonomi yang hilang. Implementasi fungsi ini dapat dilihat pada gambar 3.30 dibawah.

```python
def _fill_wrong_taxonomy(self, data):
  target_rows = self.helper.get_rows_missing_values(data, self.get_taxonomy_list())
  target_cols = [data.columns.get_loc(c) for c in self.helper.get_taxonomy_cols()]
  data.iloc[target_rows, target_cols] = ''
  return data
```

Gambar 3.30. Fungsi *fill_wrong_taxonomy*() pada LAM

- *get_predictions*(), Fungsi ini berfungsi untuk mendapatkan *DataFrame* berisi prediksi. Pertama, ia mengecek apakah atribut predictions sudah memiliki nilai. Jika belum, fungsi *inference* (yang kemungkinan besar merupakan fungsi untuk melakukan prediksi menggunakan model) akan dipanggil untuk mendapatkan prediksi. Hasil prediksi kemudian diubah menjadi *DataFrame* menggunakan fungsi *_arr_to_df*() dan disimpan ke dalam atribut predictions. Terakhir, fungsi ini mengembalikan *DataFrame* berisi

prediksi. Implementasi fungsi ini dapat dilihat pada gambar 3.31 dibawah.

```
def get_predictions(self, max_preds):
  #Get predictions DataFrame

  if self.predictions is not None:
    return self.predictions
  else:
    predictions = self.inference(max_preds)
    self.predictions_inference = predictions
    predictions = self._arr_to_df(predictions, max_preds)
    self.predictions = predictions
    return predictions
```

Gambar 3.31. Fungsi *get_predictions*() pada LAM

- *get_predictions_article*(), fungsi ini berfungsi untuk mendapatkan *DataFrame* yang berisi pemetaan antara artikel dan taksonomi berdasarkan ambang batas (*threshold*) tertentu. Ia menggunakan fungsi *get_predictions*() untuk mendapatkan *DataFrame* prediksi, kemudian menggabungkannya dengan *DataFrame* berisi artikel dan taksonomi. Fungsi ini kemudian memfilter prediksi berdasarkan ambang batas yang diberikan dan memilih taksonomi yang melebihi ambang batas tersebut. Terakhir, fungsi ini mengembalikan *DataFrame* yang berisi pemetaan antara artikel dan taksonomi hasil filtering. Implementasi fungsi ini dapat dilihat pada gambar 3.32 dibawah.

```python
def get_predictions_article(self, max_preds, threshold, keep_original_taxonomy=False):
  #Get Mapped DataFrame base on Threshold

  #=== Initialization Variable
  predictions = self.get_predictions(max_preds)
  article = self.get_article()
  article_merged = self.get_article_merged()
  article_key_col = self.helper.get_key_col(article_merged)
  article_subset_cols = self.helper.get_subset_cols(article_merged)

  #Concat Predictions and Article Horizontally
  predictions_article_merged = pd.concat(
      [article_merged[article_key_col], predictions], axis=1)

  #Merge Article and Taxonomy on 'merged taxonomy' Horizontally
  KEY_COL_PREDS = 'Taxonomy 1'
  TAXONOMY_COLS = self.helper.get_taxonomy_cols()

  article_taxonomy_merged = pd.merge(predictions_article_merged,
                                     self.get_taxonomy(),
                                     left_on=KEY_COL_PREDS,
                                     right_on=self.get_taxonomy_label(),
                                     how='left')

  # article_taxonomy_merged = self._fill_wrong_taxonomy(article_taxonomy_merged)
  article_taxonomy_merged.loc[
      article_taxonomy_merged['Score 1'] <= threshold, TAXONOMY_COLS] = ''

  #Merge subset of Article and predictions on Key Article Key Columns Horizontally
  article_subset = article[article_subset_cols] if keep_original_taxonomy else article
  predictions_subset = article_taxonomy_merged[[article_key_col] + TAXONOMY_COLS]
  predictions_article = pd.merge(article_subset, predictions_subset,
                                 on=article_key_col, how='inner')

  return predictions_article
```

Gambar 3.32. Fungsi *get_predictions_article*() pada LAM

- *_export_prediction*(), fungsi ini berfungsi untuk mengekspor hasil prediksi ke *file* Excel. Ia menerima argumen *export_path*() yang berisi path ke *file* Excel, predictions yang berisi *DataFrame* prediksi, predictions_article() yang berisi DataFrame pemetaan artikel dan taksonomi, dan *article_negative_pair*() yang berisi DataFrame artikel dengan pasangan negatifnya. Fungsi ini menuliskan ketiga *DataFrame* tersebut ke sheet yang berbeda di dalam file Excel yang sama. Implementasi fungsi ini dapat dilihat pada gambar 3.33 dibawah.

```
def _export_prediction(self, export_path, predictions,
                       predictions_article, article_negative_pair):
  with pd.ExcelWriter(export_path) as writer:

    if article_negative_pair is not None:
      article_negative_pair.to_excel(writer,
                                     sheet_name='Original With Negative Pair',
                                     index=False, na_rep='')

    predictions_article.to_excel(writer, sheet_name='Mapping', index=False, na_rep='')
    predictions.to_excel(writer, sheet_name='Predictions', index=False, na_rep='')

  print(f"File succesfully exported to {export_path}")
```

Gambar 3.33. Fungsi *_export_prediction*() pada LAM

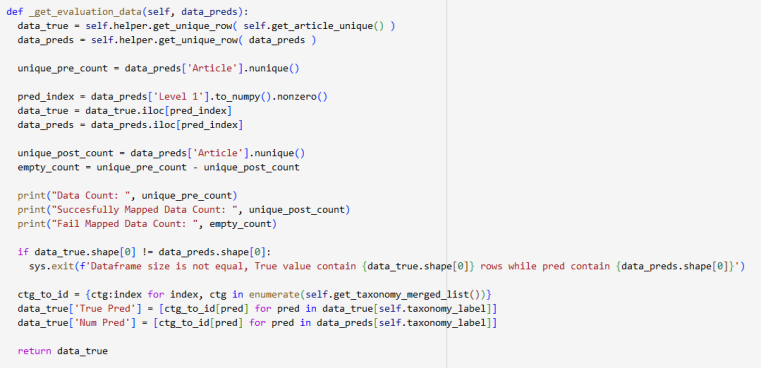
- *predict*(), fungsi ini merupakan fungsi utama untuk memprediksi taksonomi dari sebuah artikel. Ia menggunakan fungsi *get_predictions*() dan *get_predictions_article*() untuk mendapatkan *DataFrame* prediksi dan pemetaan artikel dan taksonomi. Fungsi ini juga dapat menghasilkan *DataFrame* berisi pasangan artikel dengan prediksi negatif (apabila pasangan negatif tersedia). Terakhir, fungsi ini memanggil fungsi *_export_prediction*() untuk mengekspor hasil prediksi ke file Excel. Implementasi fungsi ini dapat dilihat pada gambar 3.34 dibawah.

```
def predict(self, export_path, keep_original_taxonomy=False, max_preds=5, threshold=0.7, is_negative_pair=True):
  predictions = self.get_predictions(max_preds)
  predictions_article = self.get_predictions_article(max_preds,
                                                     threshold,
                                                     keep_original_taxonomy)

  article_negative_pair = None
  if self.helper.cols_exist(self.get_article_merged(),
                            self.get_taxonomy_label()) and is_negative_pair:
    article_negative_pair = self.helper.get_negative_pair(
        self.get_article_merged(), predictions, self.get_taxonomy_label())

  article_negative_pair_copy = article_negative_pair.copy() if article_negative_pair is not None else article_negative_pair
  self._export_prediction(export_path, predictions.copy(), predictions_article.copy(), article_negative_pair_copy)

  return predictions_article, predictions, article_negative_pair
```

Gambar 3.34. Fungsi *predict*() pada LAM

- *_get_evaluation_data*(), fungsi ini berfungsi untuk menyiapkan data yang diperlukan untuk evaluasi. Ia menerima argumen *data_preds* yang berisi *DataFrame* prediksi. Fungsi ini kemudian mengambil data kebenaran (*ground truth*) dari taksonomi artikel dan menyesuaikannya

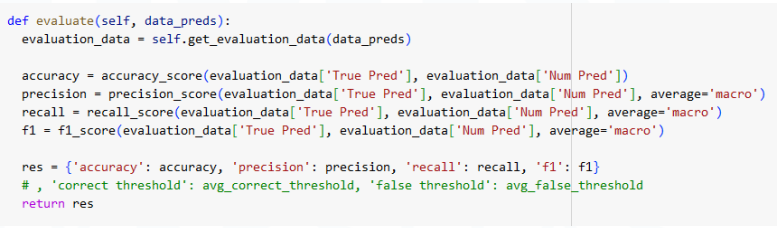
dengan format *DataFrame* prediksi. Terakhir, fungsi ini mengembalikan DataFrame yang berisi data untuk evaluasi. Implementasi fungsi ini dapat dilihat pada gambar 3.35 dibawah.



```
def _get_evaluation_data(self, data_preds):
  data_true = self.helper.get_unique_row( self.get_article_unique() )
  data_preds = self.helper.get_unique_row( data_preds )

  unique_pre_count = data_preds['Article'].nunique()

  pred_index = data_preds['Level 1'].to_numpy().nonzero()
  data_true = data_true.iloc[pred_index]
  data_preds = data_preds.iloc[pred_index]

  unique_post_count = data_preds['Article'].nunique()
  empty_count = unique_pre_count - unique_post_count

  print("Data Count: ", unique_pre_count)
  print("Succesfully Mapped Data Count: ", unique_post_count)
  print("Fail Mapped Data Count: ", empty_count)

  if data_true.shape[0] != data_preds.shape[0]:
    sys.exit(f'Dataframe size is not equal, True value contain {data_true.shape[0]} rows while pred contain {data_preds.shape[0]}')

  ctg_to_id = {ctg:index for index, ctg in enumerate(self.get_taxonomy_merged_list())}
  data_true['True Pred'] = [ctg_to_id[pred] for pred in data_true[self.taxonomy_label]]
  data_true['Num Pred'] = [ctg_to_id[pred] for pred in data_preds[self.taxonomy_label]]

  return data_true
```

Gambar 3.35. Fungsi *_get_evaluation_data*() pada LAM

- *evaluate*(), fungsi ini menghitung metrik evaluasi seperti akurasi, presisi, recall, dan F1-*Score* dari data prediksi. Ia menggunakan fungsi *_get_evaluation_data*() untuk menyiapkan data evaluasi, kemudian menghitung metrik evaluasi menggunakan library Scikit-learn. Terakhir, fungsi ini mengembalikan *dictionary* yang berisi nilai metrik evaluasi. Implementasi fungsi ini dapat dilihat pada gambar 3.36 dibawah.



```
def evaluate(self, data_preds):
  evaluation_data = self.get_evaluation_data(data_preds)

  accuracy = accuracy_score(evaluation_data['True Pred'], evaluation_data['Num Pred'])
  precision = precision_score(evaluation_data['True Pred'], evaluation_data['Num Pred'], average='macro')
  recall = recall_score(evaluation_data['True Pred'], evaluation_data['Num Pred'], average='macro')
  f1 = f1_score(evaluation_data['True Pred'], evaluation_data['Num Pred'], average='macro')

  res = {'accuracy': accuracy, 'precision': precision, 'recall': recall, 'f1': f1}
  # , 'correct threshold': avg_correct_threshold, 'false threshold': avg_false_threshold
  return res
```

Gambar 3.36. Fungsi *evaluate*() pada LAM

- *macro_evaluate*(), fungsi ini menghitung akurasi untuk setiap level taksonomi. Ia mengelompokkan data evaluasi

berdasarkan level taksonomi pertama, kemudian menghitung akurasi untuk setiap grup. Fungsi ini berguna untuk melihat performa model pada setiap level taksonomi secara terpisah. Implementasi fungsi ini dapat dilihat pada gambar 3.37 dibawah.

```
def macro_evaluate(self, data_preds):
  evaluation_data = self.get_evaluation_data(data_preds)

  data_grouped = evaluation_data.groupby('Level 1')
  accuracy_per_group = data_grouped.apply(lambda x: accuracy_score(x['True Pred'],
                                                                   x['Num Pred']))
  return accuracy_per_group
```

Gambar 3.37. Fungsi *macro_evaluate*() pada LAM

d. ***Stable Release***

Tahapan ini dilakukan setelah proses segmentasi berbasis *human-in-the-loop* (proses segmentasi yang menggabungkan model berbasis *machine learning* dengan manusia sebagai *validator*) dilakukan pada seluruh data artikel tahap pertama. Berlangsungnya proses segmentasi hingga penerapan projek *Product Information Management* (PIM) terjadi pada seluruh *business units* yang berada dibawah Kawan Lama Group, maka dari itu pengembangan performa segmentasi model melalui peningkatan data latih perlu dilakukan. Peningkatan data latih diawali dengan perbandingan beberapa model *pre-trained* dalam rangka mencari model dasar yang mampu meningkatkan akurasi segmentasi. Proses eksperimentasi ini membandingkan 2 model yang ditampilkan pada tabel 3.4. dibawah. Model menggunakan pengaturan yang serupa dengan *fine-tuning* pada model NMR-V0.1 yakni *epoch* sebesar 6, dan *batch size* 32. Perbandingan kedua model dibawah menunjukan model ‘sup-simcse-roberta-large’ mampu menghasilkan akurasi 1.68% lebih baik dibandingkan dengan ‘ALL-MPNET-BASE-V2’.

Tabel 3.4. Perbandingan Akurasi Model *Pre-trained*

Model *Pre-trained*	Akurasi@1	Presisi@1
princeton-nlp/SUP-SIMCSE-ROBERTA-LARGE	81.13%	81.13%
sentence-transformers/ ALL-MPNET-BASE-V2	79.45%	79.45%

Memahami hal tersebut, maka proses *fine-tuning* dengan peningkatan data latih dilakukan dengan menggunakan model *pre-trained* 'SUP-SIMCSE-ROBERTA-LARGE'. Secara total 350.000 baris data latih digunakan pada proses ini, dengan konfigurasi 6 *epoch* dan penurunan *batch size* menjadi 64, penurunan ini bertujuan untuk mengoptimalkan proses komputasi selama proses *fine-tuning* berlangsung. Model ini dinamakan 'CSE-NMR-V.1'. Tabel 3.5. dibawah menjabarkan hasil *fine-tuning* model 'CSE-NMR-V.1' per *epoch*.

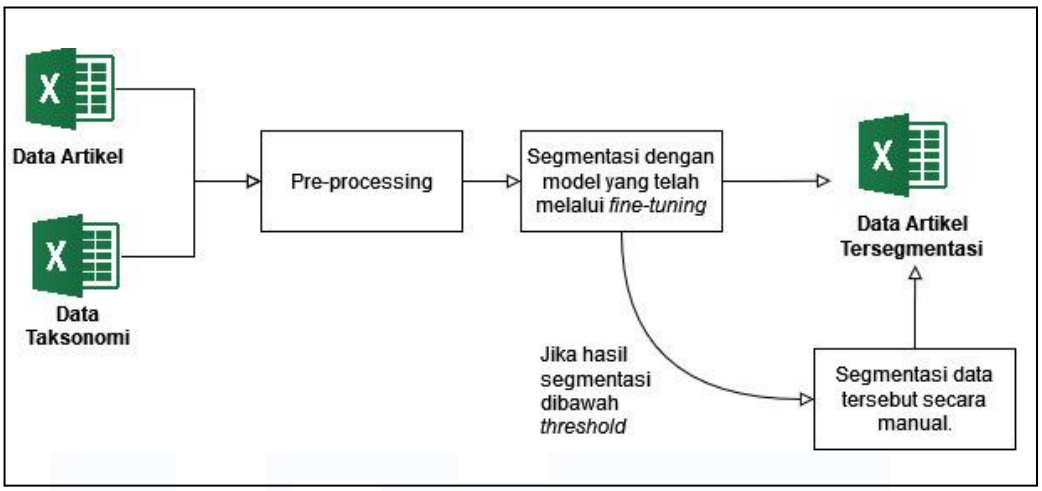
Tabel 3.5. Hasil *Fine-tuning* CSE-NMR-V.1 per *Epoch*

Epoch	**Akurasi@1**	**Presisi@1**
1	69.93%	69.93%
2	73.11%	73.11%
3	76.64%	76.64%
4	79.69%	79.69%
5	82.74%	82.74%
6	85.21%	85.21%

Memahami tabel diatas dapat dipahami bahwa peningkatan data latih mampu meningkatkan performa model CSE-NMR-V.1 sebesar 5.76% dibandingkan dengan pendahulunya NMR-V0.1. Peningkatan performa ini menunjukan generalitas yang dimiliki oleh model. Bersamaan dengan persetujuan *supervisor*, model CSE-NMR-V.1 memasuki tahapan *stable release,* menandakan kesiapan model untuk digunakan pada proses segmentasi berbasis *human-in-the-loop* kedepannya diiringi dengan *maintenance* serta penambahan data latih setiap berakhirnya proses segmentasi dimasa yang akan mendatang.

3.2.5 Pengembangan Antarmuka *Human-in-the-Loop*

Setelah *initial release* model NMR-V0.1. dilakukan, maka model berbasis *deep learning* SBERT yang telah melalui *fine-tuning* dapat digunakan pada proses segmentasi. Namun, model ini masih memiliki persentase kesalahan (*error rate*) sebesar 21.55%. Hal ini berarti untuk setiap 1000 data artikel yang disegmentasi, 215 diantaranya memiliki kecenderungan untuk masuk ke dalam segmentasi produk yang salah. Memperhatikan tujuan daripada implementasi *Product Information Management* (PIM) untuk menyajikan analitik data dalam skala korporat, menjadikan kualitas dan akurasi data sebagai faktor utama yang perlu diperhatikan dalam proses segmentasi. Memahami kedua hal tersebut, maka pendekatan *human-in-the-loop* seperti pada diagram pada gambar 3.38 dibawah perlu dilakukan. Pendekatan ini mampu meningkatkan produktivitas serta efektivitas proses segmentasi dengan mengutilisasi model NMR-V0.1, sekaligus mempertahankan kualitas serta akurasi dari segmentasi melalui validasi data hasil segmentasi model serta segmentasi secara manual untuk data artikel yang gagal tersegmentasi oleh model (tidak melewati *threshold*).



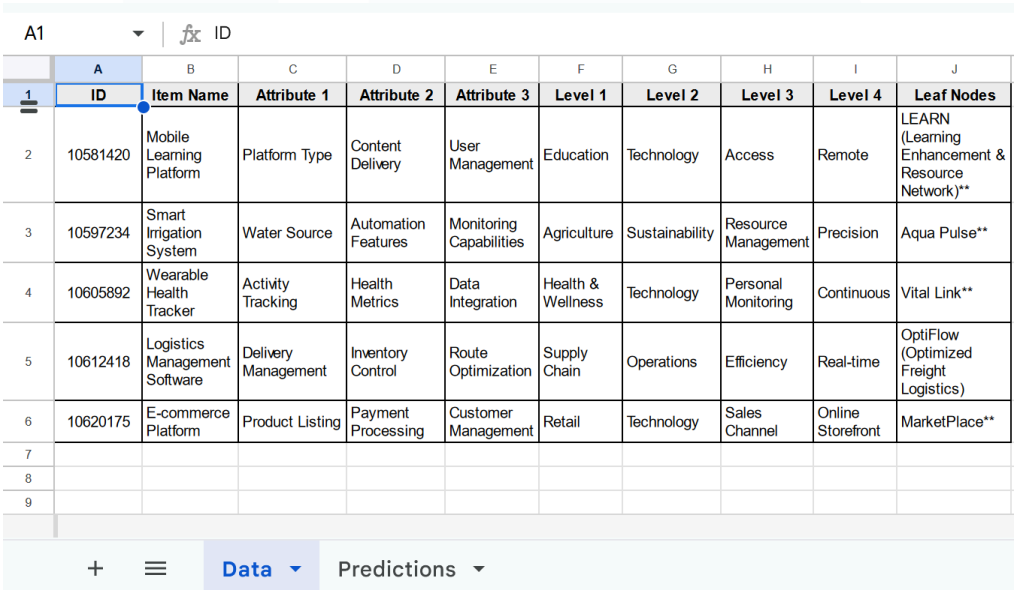
Gambar 3.38. Diagram Proses Segmentasi dengan *human-in-the-loop*

Berdasar kepada diagram diatas proses segmentasi *human-the-loop* dimulai dengan melakukan *pre-processing* pada dua data: data artikel yang menjadi target segmentasi serta data taksonomi. Selanjutnya, proses segmentasi menggunakan model yang telah melalui proses *fine-tuning* sebelumnya 'NMR-V1.0' atau 'CSE-NMR-V1.0' dilakukan. Untuk mempertahankan kualitas data segmentasi, data dengan *threshold* tertentu. SBERT sebagai fondasi awal *fine-tuning* dilakukan model *deep learning* sehingga membutuhkan GPU dalam inferensinya, hal ini karena penggunaan GPU memungkinkan komputasi secara paralel yang berdampak pada kecepatan proses segmentasi. Pemuatan data, *pre-processing* pada data, hingga proses segmentasi semuanya telah dimodularisasi kedalam sebuah *class* yang bernama LAM. Inisiasi *class* serta pemanggilan fungsi prediksi untuk melakukan segmentasi dilakukan Colab seperti pada gambar 3.39 dibawah. Pada projek ini angka 70% untuk *threshold* digunakan, sesuai dengan kesepakatan yang telah ditentukan oleh *supervisor*.

```
[ ] LAM_NMR_V1_CSE = LAM(LAM_NMR_V1_CSE_PATH, mapping_mar, taxonomy, ARTICLE_LABEL, TAXONOMY_LABEL)

    EXPORT_PATH = '/content/drive/MyDrive/Lexicon/Lexicon_auto_mapper/Data/has_mapped/Mapping Mar Extend.xlsx'
    predicitons_article, predictions, pred_negative_pair = LAM_NMR_V1_CSE.predict(EXPORT_PATH, is_negative_pair=False, threshold=0.6)

    Running Inference on:  /content/drive/MyDrive/Lexicon/Lexicon_auto_mapper/checkpoints/lam-nmr-v1-cse
    File succesfully exported to /content/drive/MyDrive/Lexicon/Lexicon_auto_mapper/Data/has_mapped/Mapping Mar Extend.xlsx
```

Gambar 3.39. Proses Inisasi *Class* serta Pemanggilan Fungsi Prediksi

Setelah segmentasi berhasil dilakukan, model akan menghasilkan keluaran berupa Excel dengan dua *sheet* yakni data artikel yang telah tersegmentasi (gambar dengan nomor satu) dan 5 data hasil prediksi beserta taksonominya (gambar dengan nomor dua) seperti pada gambar 3.40 dibawah.



A1 | ID

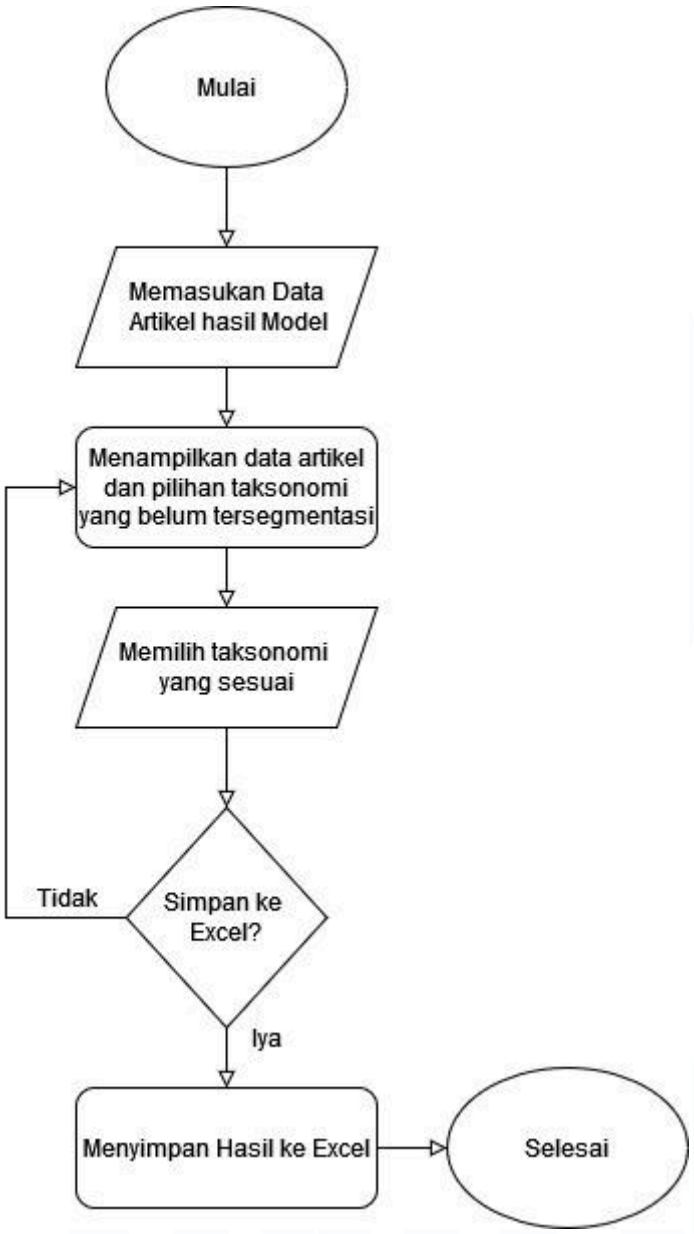
	A	B	C	D	E	F	G	H	I	J
1	ID	Item Name	Attribute 1	Attribute 2	Attribute 3	Level 1	Level 2	Level 3	Level 4	Leaf Nodes
2	10581420	Mobile Learning Platform	Platform Type	Content Delivery	User Management	Education	Technology	Access	Remote	LEARN (Learning Enhancement & Resource Network)**
3	10597234	Smart Irrigation System	Water Source	Automation Features	Monitoring Capabilities	Agriculture	Sustainability	Resource Management	Precision	Aqua Pulse**
4	10605892	Wearable Health Tracker	Activity Tracking	Health Metrics	Data Integration	Health & Wellness	Technology	Personal Monitoring	Continuous	Vital Link**
5	10612418	Logistics Management Software	Delivery Management	Inventory Control	Route Optimization	Supply Chain	Operations	Efficiency	Real-time	OptiFlow (Optimized Freight Logistics)
6	10620175	E-commerce Platform	Product Listing	Payment Processing	Customer Management	Retail	Technology	Sales Channel	Online Storefront	MarketPlace**
7										
8										
9										

Gambar 3.40. Excel Hasil Prediksi dengan Dua *Sheet*

Tantangan terbesar dalam melakukan segmentasi pada data artikel adalah proses pencarian taksonomi serta pemindahan taksonomi dari sumber data ke dalam Spreadsheet data yang menjadi target segmentasi. Memahami hal tersebut maka pengembangan antarmuka *human-in-the-loop* dilakukan untuk mengurangi proses tersebut hanya melalui satu antarmuka dengan 5 pilihan taksonomi yang tidak berhasil melewati *threshold* serta kebebasan untuk memilih taksonomi lainnya yang lebih sesuai. Penggunaan antarmuka mampu mengurangi *human error* saat pemindahan taksonomi, dengan tetap mempertahankan efektivitas proses. Gambar 3.41. dibawah menampilkan antarmuka *human-in-the-loop.*

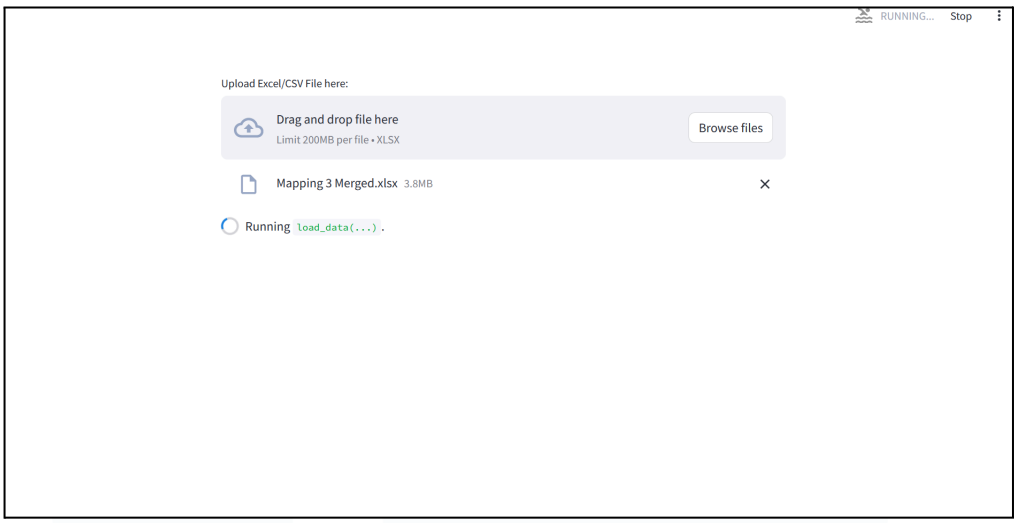
Gambar 3.41. Antarmuka *Human-in-the-loop*

Antarmuka ini dikembangkan dengan menggunakan Colab untuk meningkatkan aksesibilitas bagi seluruh tim MDM *data analyst intern* untuk digunakan, Streamlit sebagai *library* berbasis python yang umum digunakan untuk pengembangan antarmuka website, dan Tornado untuk menjalankan *local server*. Diagram pada gambar 3.42. dibawah menjelaskan bagaimana cara kerja dari antarmuka dalam melakukan segmentasi data.

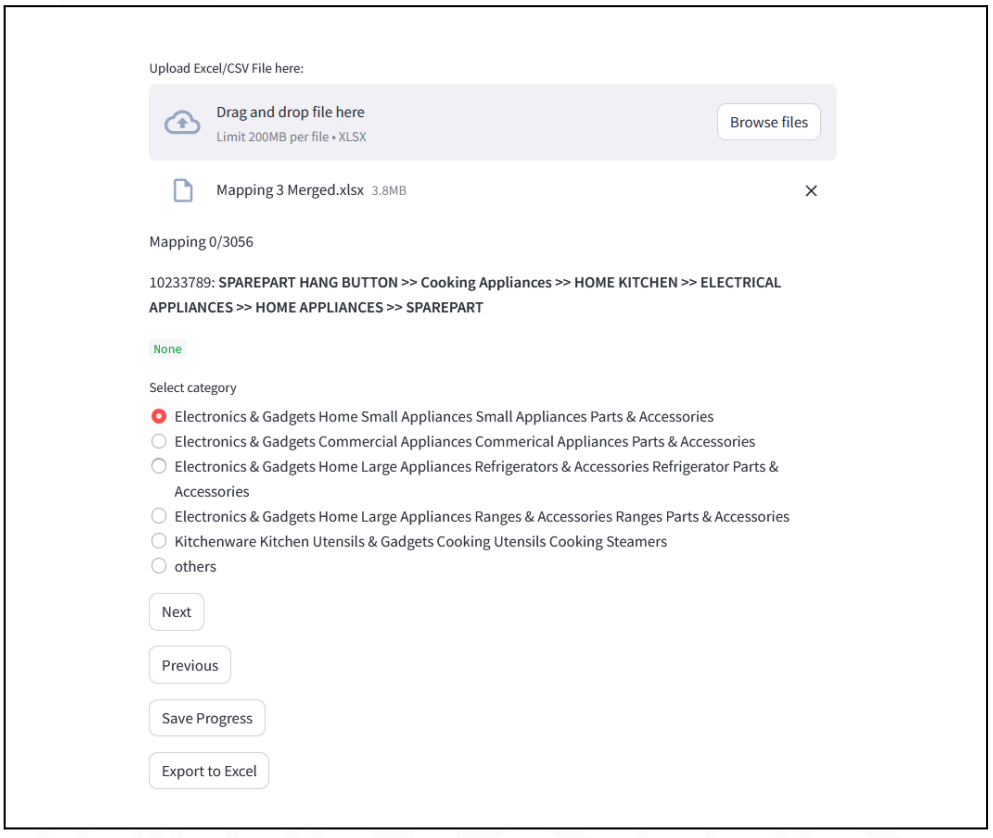


Gambar 3.42. *Flowchart* Antarmuka *Human-in-the-loop*

Berdasarkan *flowchart* diatas, sistem dimulai dengan pengguna memasukkan *file* Excel yang berisi dua *sheet* (data artikel yang telah tersegmentasi dan hasil prediksi) yang telah melalui proses segmentasi sebelumnya. Proses memasukan *file* dilakukan seperti pada gambar 3.43. dibawah.



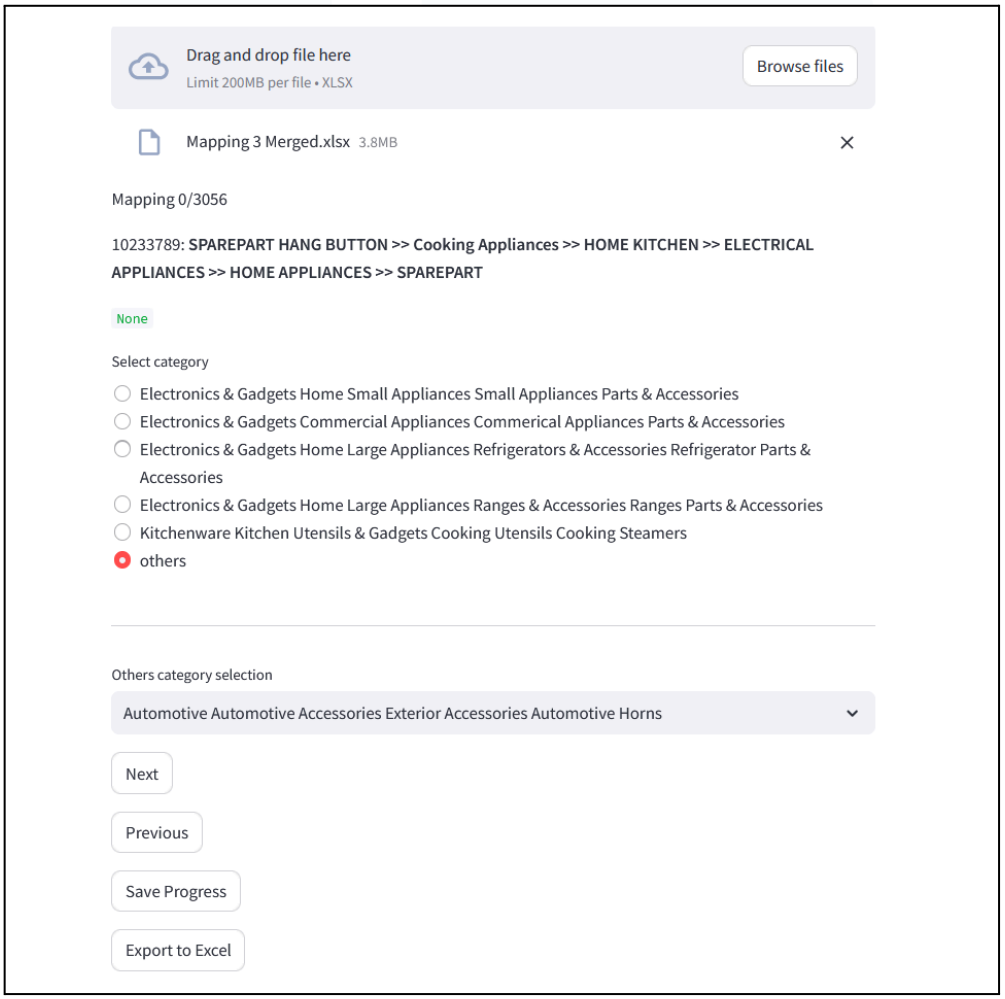
Gambar 3.43. Memasukan *File* Excel kedalam Antarmuka



Gambar 3.44. Tampilan Antarmuka setelah Data Diunggah

Kemudian, sistem akan memuat data tersebut dan menampilkan artikel dengan taksonomi kosong, beserta dengan pilihan taksonomi sesuai pada *sheet* hasil prediksi yang dapat dipilih oleh pengguna. Adapun tampilan daripada

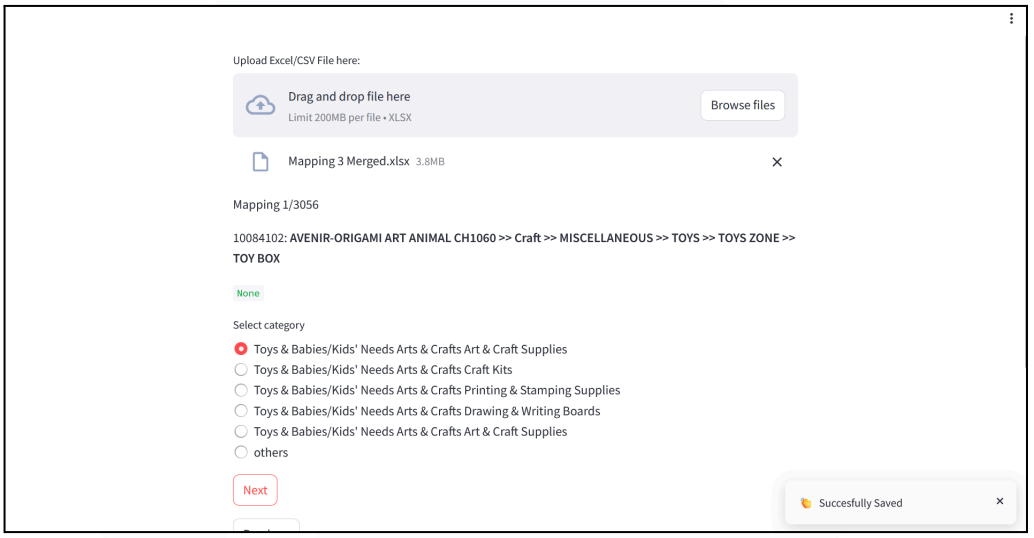
antarmuka *human-in-the-loop* setelah data berhasil diunggah dapat dilihat pada gambar 3.44 diatas. Jika diantara 5 pilihan tidak ada yang secara tepat menggambarkan sebuah produk, pengguna dapat memilih opsi *others* untuk memilih opsi lainnya sesuai dengan pilihan taksonomi yang ada pada Kawan Lama Group. Tampilan jika pengguna memilih *others* dapat dilihat pada gambar 3.45. dibawah.



Gambar 3.45. Tampilan Antarmuka Jika Memilih *Others*

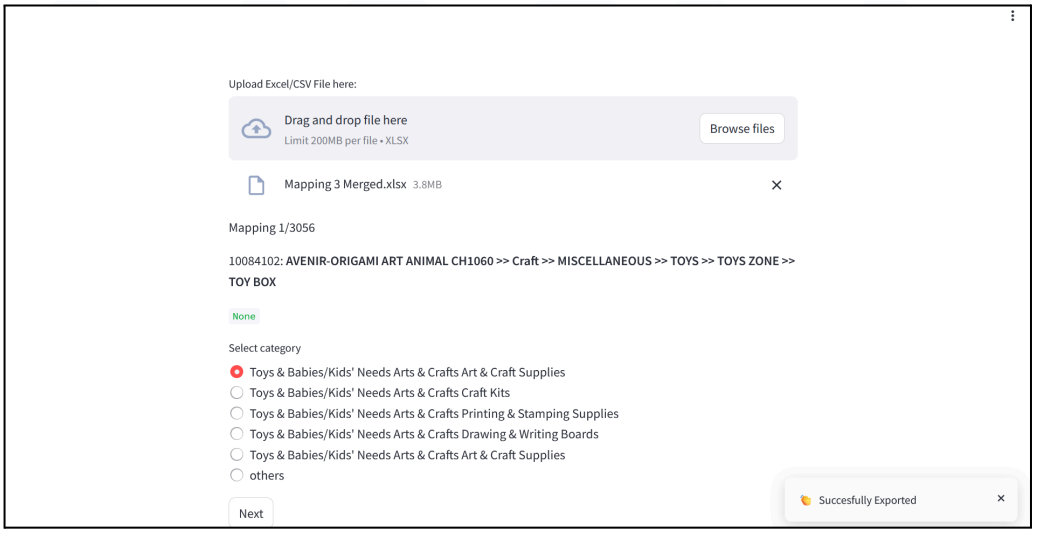
Setelah segmentasi berhasil dilakukan untuk sebuah artikel, sistem secara otomatis menyimpan progres tersebut. Kemudian, pengguna dapat memilih untuk melanjutkan ke artikel berikutnya atau kembali ke artikel sebelumnya jika perlu untuk melakukan koreksi atau penyesuaian. Setiap kali pengguna

memilih kategori untuk artikel, progresnya akan disimpan oleh sistem. *Pop-up notification* juga akan tampil seperti pada gambar 3.46. dibawah.



Gambar 3.46. Tampilan Antarmuka dengan Notifikasi Tersimpan

Setelah semua artikel ditugaskan kategorinya, pengguna memiliki opsi untuk mengekspor hasil akhir ke dalam file Excel. *File* Excel akan tersimpan pada lokal Colab, dilengkapi pula dengan notifikasi seperti yang ditampilkan pada gambar 3.47. dibawah.



Gambar 3.47. Tampilan Antarmuka Setelah Menyimpan *File* Excel

Proses ini memberikan fleksibilitas kepada pengguna untuk secara interaktif menugaskan kategori untuk setiap artikel dalam dataset, sambil memberikan kemampuan untuk menyimpan progres atau mengekspor hasil kapan saja. Alur kerja sistem ini dapat direpresentasikan sebagai alur langkah-langkah yang berurutan, dimulai dari memuat data, menampilkan artikel, memilih kategori, menyimpan progres, dan akhirnya mengekspor hasil akhir.

3.2.6 Segmentasi Data dengan pendekatan *Human-in-the-Loop*

Pasca terbentuknya model NMR-V0.1 dan antarmuka penunjang, proses segmentasi beralih dari yang sebelumnya dilakukan secara manual kini menjadi *hybrid*. Proses *hybrid* dengan melibatkan model dalam melakukan segmentasi serta manusia dalam verifikasi dikenal sebagai *human-in-the-loop*. Segmentasi dengan pendekatan *human-in-the-loop* terbagi kedalam tiga tahapan:

a. Tahap pertama

Segmentasi tahap pertama berfokus pada segmentasi data artikel yang belum terselesaikan secara manual sebelum nantinya digunakan untuk meningkatkan kemampuan data. 1/7 dari total keseluruhan data artikel telah terselesaikan secara manual, menyisakan 6/7 untuk disegmentasi atau 300.000 baris data. Produktivitas serta efektivitas daripada penerapan *human-in-the-loop* dalam segmentasi data artikel dapat dilihat pada tabel 3.6. dibawah.

Tabel 3.6. Perbandingan Proses Manual dengan *Human-in-the-loop*

	Manual	*Human-in-the-loop*
Artikel/orang/hari	833	1500

Durasi/artikel	34.2 detik	19.2 detik

Berdasarkan tabel di atas, implementasi segmentasi dengan pendekatan *human-in-the-loop* telah berhasil meningkatkan produktivitas proses segmentasi hampir dua kali lipat. Sebelumnya, proses segmentasi 50.000 baris data yang dikerjakan oleh tiga orang memakan waktu selama satu bulan. Namun, dengan adanya pendekatan *human-in-the-loop* dan penambahan dua anggota tim, total lima orang dapat mempercepat proses segmentasi 300.000 baris data hanya dalam waktu 8 minggu. Hal ini menunjukkan peningkatan yang signifikan dalam efisiensi dan kecepatan penyelesaian tugas segmentasi dengan melibatkan lebih banyak anggota tim serta implementasi *human-in-the-loop*.

b. Tahap kedua

Segmentasi data artikel tahap kedua merupakan kelanjutan dari tahap pertama yang bersifat tambahan (*extend*). Proses segmentasi ini mengimplementasi model CSE-NMR-V.1, model yang telah dilatih pada seluruh data artikel yang telah segmentasi dan tervalidasi pada tahap pertama. Tahapan kedua berlangsung dalam periode satu minggu, dengan total 35.000 baris data artikel.

c. Tahap ketiga

Segmentasi data artikel terakhir pada periode magang dilakukan pasca berakhirnya implementasi *Project Information Management* (PIM) pada *business unit* industrial Kawan Lama Group. Pada tahapan ini secara total 40.695 baris data akan melalui proses segmentasi. Proses ini berlangsung selama periode 2 minggu.

Penerapan pendekatan *human-in-the-loop* dalam proses segmentasi data artikel secara kuantitatif mampu memberikan dampak positif yang

sangat signifikan terhadap efisiensi dan produktivitas dengan menggabungkan model dan intervensi manusia. Peningkatan jumlah artikel yang dapat diselesaikan per orang per hari serta durasi yang lebih singkat untuk menyelesaikan satu artikel. Pendekatan ini tidak hanya meningkatkan efisiensi proses, tetapi juga mengoptimalkan kualitas segmentasi data secara keseluruhan. Dengan penerapan pendekatan *human-in-the-loop*, diharapkan usaha terus dilakukan untuk meningkatkan kinerja dan ketepatan dalam pengelolaan data secara efektif.

3.2.7 Validasi Artikel pasca Segmentasi

Tahapan terakhir pasca proses segmentasi baik yang dilakukan secara manual maupun dengan proses *human-in-the-loop* adalah validasi. Tahapan validasi terbagi kedalam beberapa tahap:

a. Pemisahan data yang tidak dapat tersegmentasi

   Saat ini, data taksonomi yang tersedia belum mencakup seluruh produk yang dimiliki oleh Kawan Lama Group. Oleh karena itu, diperlukan pemisahan data untuk mengelompokkan artikel-artikel yang belum berhasil tersegmentasi. Langkah ini penting guna memastikan bahwa semua produk yang dimiliki oleh Kawan Lama Group dapat diidentifikasi dan dimasukkan ke dalam kategori yang sesuai berdasarkan karakteristiknya. Dengan melakukan pemisahan data ini secara cermat dan terstruktur, akan memungkinkan untuk memperluas cakupan taksonomi dan meningkatkan ketepatan dalam penentuan kategori produk.

b. Pengecekan kesesuaian taksonomi

   Setelah dilakukan pemisahan data yang gagal tersegmentasi, dilakukan pengecekan manual terhadap kesesuaian taksonomi guna menjaga kualitas data. Tantangan yang dihadapi adalah jumlah data

artikel yang besar, sehingga proses pengecekan membutuhkan waktu dan usaha yang signifikan. Untuk mempercepat validasi, digunakan pivot table yang membandingkan atribut data artikel dengan kolom taksonomi. Pendekatan ini didasarkan pada asumsi bahwa atribut-atribut tertentu dapat merepresentasikan suatu taksonomi, sehingga memudahkan identifikasi kesesuaian antara data artikel dan kategori taksonomi yang telah ditetapkan. Dengan menggunakan pivot table secara efektif, diharapkan proses validasi dapat dilakukan lebih efisien dan akurat, sehingga memastikan kualitas dan ketepatan dalam pengelompokan produk berdasarkan taksonomi yang telah ditetapkan.

c. Pengecekan duplikasi pada data

Tahapan terakhir dalam proses adalah melakukan pengecekan terhadap duplikasi pada data artikel. Pengecekan ini dilakukan dengan membandingkan ID artikel dengan taksonomi yang terkait, karena beberapa artikel mungkin memiliki ID yang sama. Tujuan dari langkah ini adalah untuk meningkatkan integritas dan konsistensi taksonomi pada data, sehingga setiap produk memiliki kategori yang tepat dan tidak terjadi redundansi atau duplikasi yang tidak diinginkan. Dengan melakukan pengecekan duplikasi secara sistematis, diharapkan data taksonomi dapat dipelihara dengan baik dan menjadi lebih akurat serta bermanfaat dalam analisis dan pengelolaan produk secara efisien.

Tahapan ini menutup proses segmentasi pada setiap tahapannya. Data artikel yang telah tersegmentasi kemudian diberikan kepada divisi *data quality* untuk kemudian diproses dan diunggah kedalam *database* internal Kawan Lama Group.

3.3 Kendala yang Ditemukan

Adapun kendala yang ditemukan selama periode kerja magang sebagai MDM *Data Analyst Intern* pada Kawan Lama Group antara lain:

a. Kurangnya Komunikasi dan Koordinasi antar Anggota Tim

   Efektivitas komunikasi dan koordinasi yang kuat adalah landasan fundamental bagi tim yang ingin meraih kesuksesan. Ketika kedua aspek ini terkendala, konsekuensi negatif pun bermunculan, menghambat produktivitas, efisiensi, dan bahkan semangat tim. Kesalahpahaman mengenai tujuan dan peran masing-masing anggota dapat menimbulkan kebingungan dan arah kerja yang tidak terfokus. Komunikasi yang tidak efektif memicu kesalahan, keterlambatan, dan potensi kerugian finansial. Anggota tim yang merasa terputus dan kurang dihargai akan memiliki motivasi yang menurun serta menghasilkan pekerjaan dengan kualitas yang lebih rendah. Kurangnya komunikasi juga dapat membuka celah bagi kesalahpahaman dan hal-hal negatif, yang pada akhirnya memicu konflik dan ketegangan di dalam tim.

b. Tidak adanya Pembagian Penugasan yang Jelas

   Efektivitas tim yang baik sangat bergantung pada pembagian tugas yang jelas. Tanpa adanya kejelasan, tim akan menghadapi berbagai tantangan terkait alokasi waktu, tenaga, dan fokus para anggotanya. Keterlibatan seorang individu dalam banyak proyek secara bersamaan dapat mengurangi produktivitas, efisiensi, dan fokus mereka. Hal ini dikarenakan individu tersebut kesulitan untuk memprioritaskan dan membagi waktunya secara efektif. Akibatnya, pengerjaan tugas menjadi tersendat-sendat, rentan kesalahan, dan berpotensi menimbulkan kelelahan (burnout) pada individu. Selain itu, kurangnya kejelasan pembagian tugas juga berdampak pada fokus dan konsentrasi. Ketika individu mengerjakan banyak tugas dalam waktu bersamaan, mereka akan kesulitan untuk

memusatkan perhatiannya pada satu hal. Akibatnya, kualitas hasil kerja menurun dan risiko terjadinya kesalahan meningkat.

c. Keterbatasan Tenaga Komputasi dalam Pelatihan Model

Keterbatasan daya komputasi, khususnya dalam bentuk unit pemrosesan grafis (GPU), menjadi tantangan tersendiri dalam pelatihan model berbasis deep learning. Model-model ini, yang memiliki arsitektur rumit dan membutuhkan banyak perhitungan, memerlukan daya pemrosesan yang substansial untuk belajar secara efektif dari kumpulan data yang besar. Keterbatasan ketersediaan GPU menghambat proses pelatihan, menghadirkan keterbatasan dalam proses iterasi selama pelatihan model.

3.4 Solusi atas Kendala yang Ditemukan

Memahami kendala yang dihadapi, solusi yang dilakukan untuk mengatasi masalah tersebut antara lain:

a. Meningkatkan Komunikasi antar anggota tim

Peningkatan komunikasi antar anggota tim dapat dicapai melalui berbagai langkah. Salah Satunya membangun saluran komunikasi yang terbuka dan rutin dalam bentuk rapat mingguan, dimana semua anggota tim dapat saling terhubung dan berbagi informasi. Selain itu, mempromosikan budaya mendengarkan yang inklusif, di mana setiap anggota tim didorong untuk berpartisipasi aktif dalam diskusi dan memberikan masukan. Mediasi yang dilakukan oleh rekan mentor dan *supervisor* turut meningkatkan kualitas komunikasi antar anggota tim.

b. Mengajukan kejelasan dalam pembagian penugasan yang merata

Pembagian penugasan yang jelas dan merata adalah kunci untuk mengoptimalkan produktivitas tim. Hal ini dapat dicapai dengan mengidentifikasi keahlian dan minat masing-masing anggota tim untuk menyesuaikan tugas dengan kemampuan mereka. Memastikan bahwa

setiap anggota tim memahami tujuan umum proyek dan peran mereka dalam mencapainya akan membantu menghindari kebingungan dan overlapping tugas. Manajemen serta monitor progres setiap anggota tim turut dapat dilakukan dengan menggunakan Google Spreadsheet.

c. Penggunaan Google Colab sebagai IDE berbasis *cloud* yang menyediakan layanan GPU gratis

   Memanfaatkan Google Colab sebagai IDE berbasis cloud adalah solusi efektif untuk mengatasi keterbatasan daya komputasi dalam pelatihan model deep learning. Layanan GPU gratis yang disediakan oleh Google Colab memungkinkan tim untuk mengakses daya pemrosesan yang diperlukan untuk melatih model-model kompleks tanpa perlu menginvestasikan dalam infrastruktur hardware mahal. Dengan memanfaatkan fitur ini Colab ini pengembangan model dan iterasi tanpa terkendala oleh keterbatasan daya komputasi lokal.