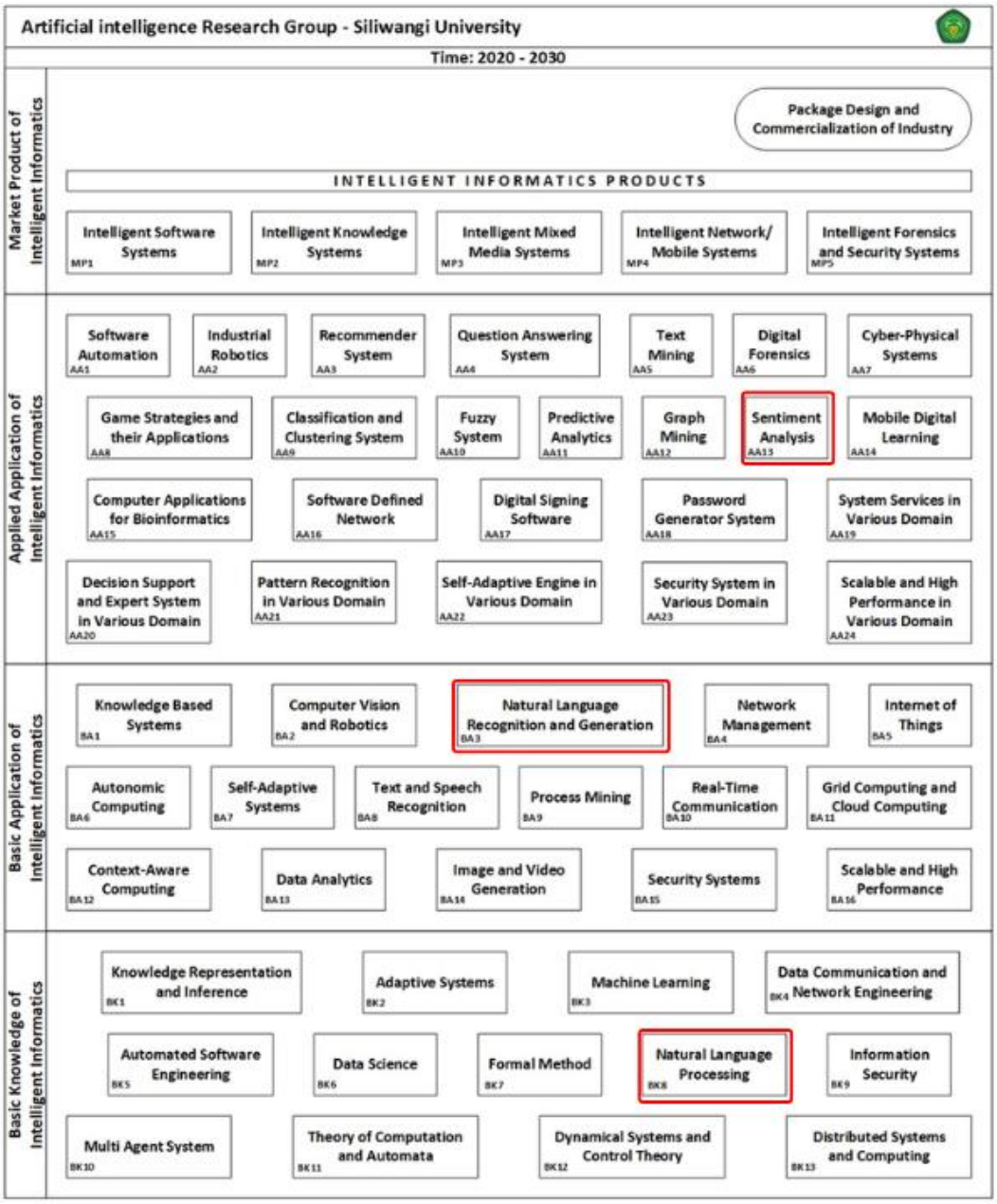
BAB III

METODOLOGI PENELITIAN

3.1. Peta Jalan (*Roadmap*) Penelitian

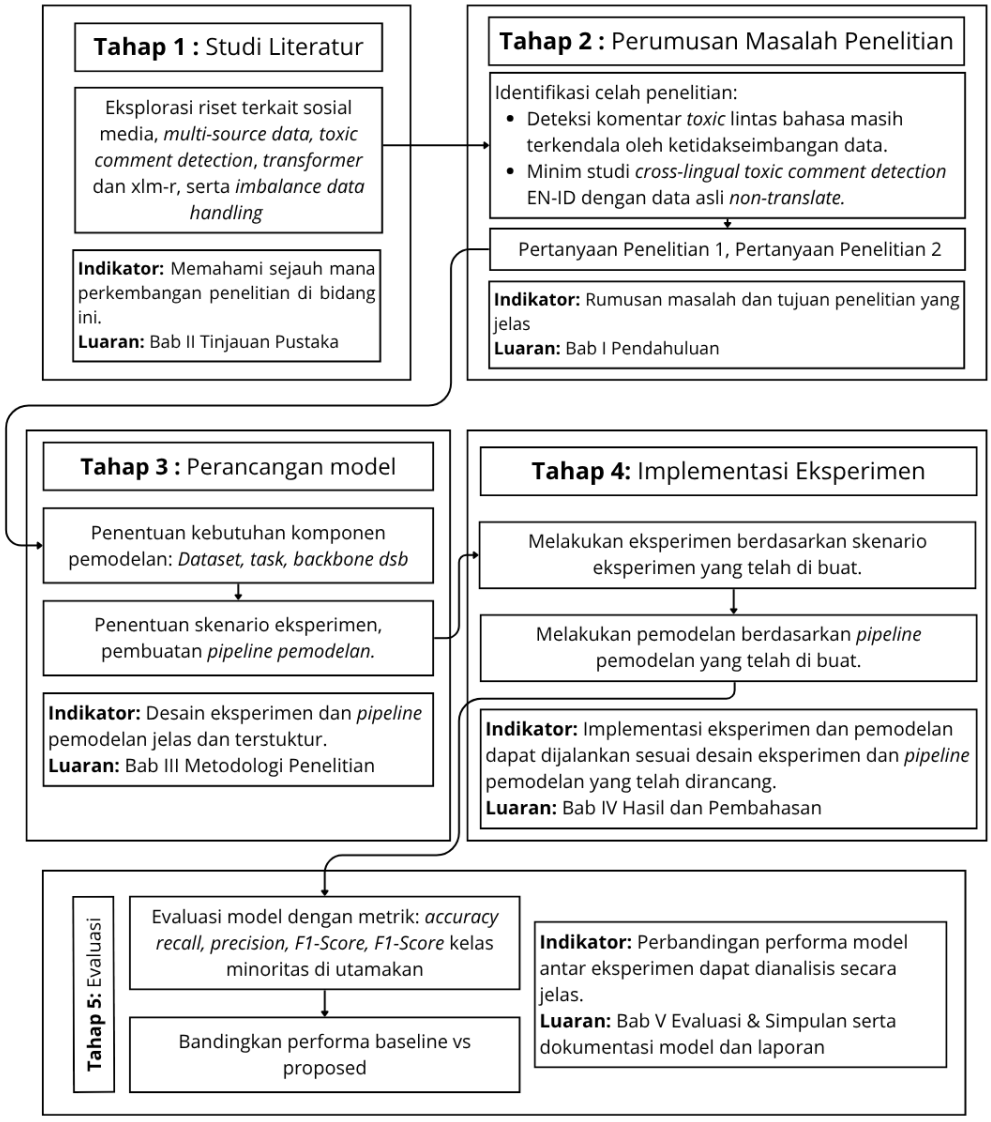


Gambar 3. 1 Peta Jalan (*Roadmap*) Penelitian

Penelitian yang diusulkan merupakan bagian dari *roadmap* penelitian Kelompok Keahlian Informatika dan Sistem Intelijen (ISI) Universitas Siliwangi. Pada Gambar 3.1 yang didapatkan dari ketua kelompok keahlian ISI menunjukkan posisi penelitian ini berada pada tiga komponen yang ditandai kotak merah, yaitu *Natural Language Processing* sebagai dasar keilmuan karena berfokus pada pemrosesan dan pemodelan bahasa alami menggunakan XLM-RoBERTa, *Natural Language Recognition and Generation* pada aspek *recognition* karena model dirancang untuk mengenali dan mengklasifikasi pola ujaran *toxic* dalam teks, serta *Sentiment Analysis* pada level aplikasi karena melakukan analisis dan klasifikasi komentar menjadi kategori *toxic* dan *non-toxic.*

3.2. Tahapan Penelitian

Tahapan penelitian pada Gambar 3.2 disusun dengan sistematis untuk menjawab rumusan masalah yang telah di tetapkan, mulai dari identifikasi celah penelitian hingga evaluasi performa model. Setiap tahap dirancang agar selaras dengan tujuan utama penelitian.



Gambar 3. 2 Tahapan Penelitian

Tahap 1 yaitu studi literatur dilakukan dengan mengeksplorasi penelitian terkait *toxic comment detection, multi-source data,* XLM-RoBERTa*,* serta strategi *imbalance data handling* untuk mengidentifikasi *research gap.* Tahapan ini bertujuan untuk memahami sejauh mana perkembangan penelitian di bidang ini.

Pada Tahap 2 yaitu perumusan masalah penelitian, dilakukan identifikasi celah penelitian berdasarkan hasil studi literatur. Celah yang ditemukan antara lain permasalahan ketidakseimbangan data pada deteksi komentar *toxic* lintas bahasa

serta kelemahan penelitian sebelumnya yang masih menggunakan dataset hasil translasi. Penelitian yang melibatkan bahasa Indonesia dalam *cross-lingual toxic comment detection* juga belum ditemukan, namun tersedia dataset berbahasa asli dengan format yang selaras dengan dataset bahasa Inggris. Berdasarkan celah tersebut kemudian dirumuskan pertanyaan penelitian yang menjadi dasar penyusunan tujuan penelitian. Tahap ini menghasilkan rumusan masalah dan tujuan penelitian yang jelas.

Pada Tahap 3 yaitu perancangan model, dilakukan pemetaan kebutuhan komponen pemodelan seperti pemilihan dataset, penentuan *task*, serta penetapan *backbone* model yang digunakan. Pada tahap ini juga dirancang desain eksperimen dan *pipeline* pemodelan agar penelitian memiliki arah yang jelas dan terstruktur. Pada tahap implementasi eksperimen, desain eksperimen dan *pipeline* pemodelan yang telah disusun pada tahap sebelumnya diimplementasikan dalam proses pelatihan model. Pada tahap terakhir yaitu evaluasi, bertujuan untuk menganalisis hasil eksperimen dengan membandingkan performa model *baseline* dan metode yang diusulkan.

Penelitian ini dirancang dalam dua tahap eksperimen. Pada *stage 1*, model XLM-RoBERTa-base dengan *partial fine-tuning* diuji menggunakan tujuh konfigurasi *loss function* dengan tujuan mengidentifikasi *loss function* yang paling optimal berdasarkan evaluasi pada *validation set*. Bobot model terbaik dari *stage 1* kemudian digunakan sebagai inisialisasi pada *stage 2*, di mana seluruh 12 layer XLM-RoBERTa dilatih ulang secara penuh menggunakan *loss* BCE untuk mengevaluasi dampak *vanilla fine-tuning* terhadap performa model akhir.

Keseluruhan skenario eksperimen yang digunakan dalam penelitian ini ditunjukkan pada Tabel 3.1.

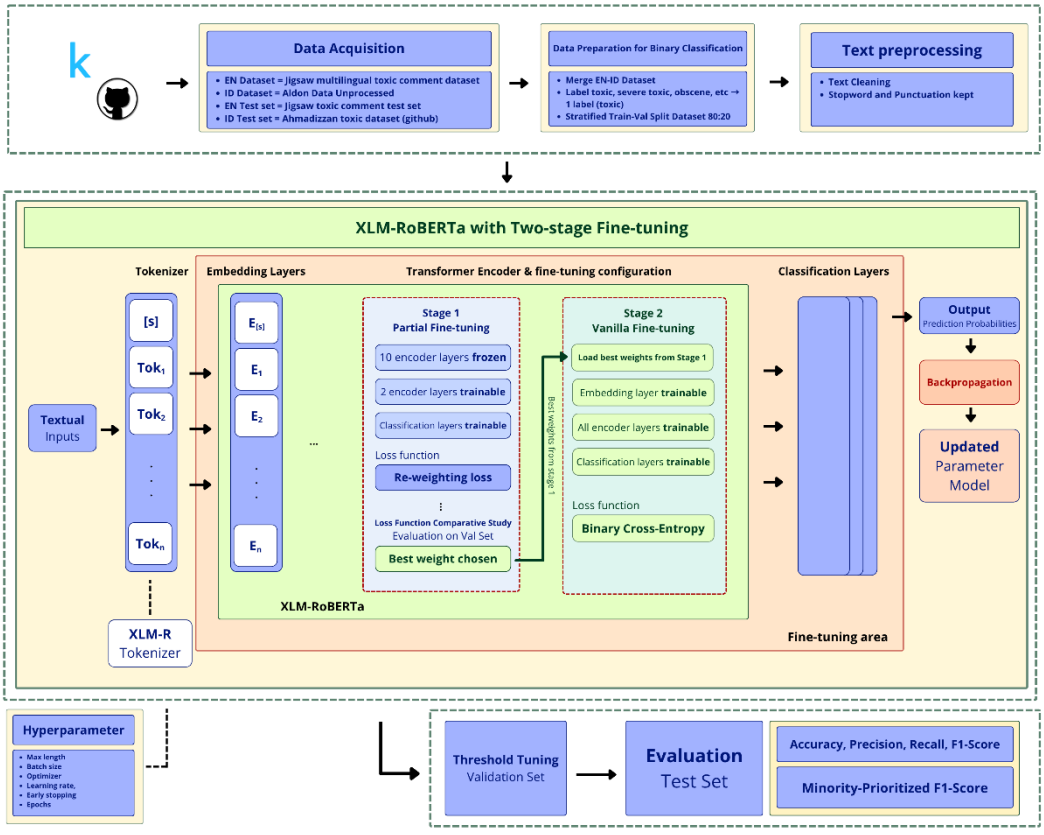
Tabel 3. 1 Skenario eksperimen

No	**Backbone**	**Layer Configuration**	**Loss Function**
Stage 1, Partial Fine-Tuning and Comparative Study			
1	Pretrained XLM-R-Base	10 frozen, 2 trainable	BCE (no imbalance data handling)
2	Pretrained XLM-R-Base	10 frozen, 2 trainable	Static Fusion BCE with Focal Loss (baseline)
3	Pretrained XLM-R-Base	10 frozen, 2 trainable	Adaptive Fusion BCE with Focal Loss
4	Pretrained XLM-R-Base	10 frozen, 2 trainable	Static Fusion BCE with DRFL
5	Pretrained XLM-R-Base	10 frozen, 2 trainable	Adaptive Fusion BCE with DRFL
Stage 2, Vanilla Fine-tuning			
6	Best weight from Stage 1	12 trainable	BCE

Studi komparatif pada tahap 1 dilakukan untuk mengidentifikasi konfigurasi *loss function* yang paling optimal dalam menangani *class imbalance* pada tugas *toxic comment detetion* lintas bahasa. Lima konfigurasi *loss function* dibandingkan mulai dari pendekatan fusi statis sampai adaptif. Seluruh eksperimen pada tahap ini menggunakan konfigurasi *backbone* dan *hyperparameter* yang identik, yaitu sepuluh *layer* dibekukan dan dua *layer* teratas dilatih, agar perbedaan performa yang teramati dapat diatribusikan secara langsung pada perbedaan *loss function* dan

bukan faktor lain. Pemilihan *loss* terbaik didasarkan pada *minority class f1-score* pada *validation set* dan bobot model terbaik tersebut selanjutnya digunakan sebagai inisialisasi pada tahap 2.

Ilustrasi *pipeline* pemodelan berbasis RoBERTa diadopsi dari penelitian sebelumnya (Yuliyanti dkk., 2026) sebagai acuan dalam merancang alur pemrosesan data dan pelatihan model. Keseluruhan *pipeline* pemodelan yang digunakan dalam penelitian ini ditunjukkan pada Gambar 3.3.



Gambar 3. 3 *Pipeline* pemodelan

Penelitian ini menggunakan data sekunder yang berasal dari berbagai platform (*multi-source*) yang merepresentasikan interaksi sosial digital. Kedua dataset pelatihan yang digunakan memiliki karakteristik skema pelabelan yang

serupa antara data berbahasa Inggris dan bahasa Indonesia. Data tersebut diperoleh dari dataset penelitian terdahulu yang tersedia secara publik pada platform Kaggle. Sementara itu, data pengujian (*test set*) diambil dari sumber yang berbeda dari dataset pelatihan, hal ini bertujuan untuk mengevaluasi kemampuan generalisasi model terhadap data OOD, sehingga performa mencerminkan kemampuan generalisasi dan adaptasi model pada data baru (Yu dkk., 2024).

Pada tahap *data preparation,* kedua dataset digabung untuk mendukung skenario *cross-lingual* bahasa Inggris dan bahasa Indonesia. Selanjutnya, seluruh kategori toksisitas yang semula terdiri dari beberapa subkelas dikonsolidasikan menjadi satu label *toxic* untuk membentuk skema *binary classification*, sebagaimana digunakan pada penelitian sebelumnya (Song, Huang, & Xiao, 2021). Pembagian data dilakukan menggunakan *stratified train-val splitting* dengan rasio 80:20 berdasarkan kombinasi bahasa dan label yang di adopsi dari penelitian sebelumnya (Sushma dkk., 2025). Teknik ini memastikan setiap kombinasi kelas tetap terwakili secara proporsional pada data pelatihan dan validasi.

Tahap *text preprocessing* bertujuan untuk menyiapkan data teks agar sesuai dengan karakteristik *input* model (Dinarta & Wicaksana, 2025). Tahap *text preprocessing* pada penelitian ini melakukan pembersihan minimal dengan dua tahap utama yang dibutuhkan oleh *tokenizer* transformer karena pembersihan yang terlalu agresif seperti penghapusan *stopword* ataupun tanda baca berpotensi mengubah makna pada sebuah kalimat dan menurunkan kinerja model klasifikasi (Miyajiwala dkk., 2022).

Tabel 3. 2 Text preprocessing

Text Cleaning	**Stop word and Punctuation kept**
Teks dimurnikan dengan menghapus *token* yang tidak relevan atau *noise* ringan seperti karakter HTML dan simbol tertentu menggunakan ekspresi reguler, tanpa melakukan normalisasi agresif agar konteks semantik tetap terjaga.	*Stop word* dan Tanda baca tidak dihapus karena memiliki nilai semantik dan membantu model memahami emosi atau struktur kalimat dalam deteksi *toxic comment*.

Pada proses pembersihan teks komentar, setiap karakter yang tidak termasuk alfanumerik dan tanda baca (! ? . ,) dibuang. Proses ini diterapkan pada seluruh data latih, validasi dan tes. Gambaran lebih lanjut mengenai proses pembersihan teks komentar menggunakan fungsi *clean_text* dapat dilihat pada Algoritma 1.

Algoritma 1. *Text preprocessing for comment data.*

```
Input: Raw comment_text (df)
Start

1. Define Cleaning Pattern
   Define CLEAN_RE as regex pattern [^a-zA-Z0-9\s!?.,]
   to identify characters that should be removed

2. Text Cleaning Function
   Create function clean_text(text):

   a. Remove all characters matching CLEAN_RE pattern
   b. Return cleaned text

3. Apply Cleaning Process
   Apply clean_text function to each row in df[comment_text]

   for each row in df:
   df[comment_text] ← clean_text(df[comment_text])

   → Output: cleaned_comment_text

End
Output: Cleaned comment_text (df)
```

Teks yang telah dibersihkan selanjutnya diproses melalui *pipeline* pemodelan XLM-RoBERTa yang dimulai dari *tokenization* hingga prediksi akhir. Gambaran lebih lanjut mengenai proses pemodelan pada *pretrained language model* XLM-RoBERTa dapat dilihat pada Algoritma 2.

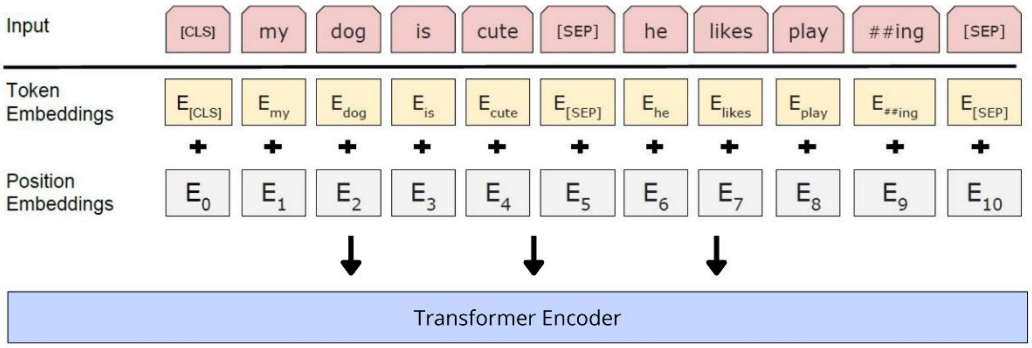
Algoritma 2. *XLM-RoBERTa.*

```
Input: Text sentence
Start

1. Tokenization
   Split text into subword token units using XLM-RoBERTa
   tokenizer and add special tokens <s> at the beginning
   and </s> at the end of the sequence
   → Output: token_ids, attention_mask

2. Embedding
   Convert token_ids into vector representations of
   dimension 768 through summation of token embedding
   and positional embedding
   → Output: input_representation

3. Transformer Encoder Processing
   Process input_representation through 12 stacked encoder
   layers using multi-head self-attention and feed-forward
   network to generate bidirectional contextual representation
   for each token
   → Output: contextual_representation

4. CLS Token Extraction
   Extract output vector of token <s> at position 0 as
   single sentence-level representation
   → Output: cls_vector (dim: 768)

5. Classification Head
   Pass cls_vector through 3 fully connected layers to
   map representation into classification space
   FC1: 768 → 1536
   FC2: 1536 → 768
   FC3: 768 → 1
   → Output: logits

6. Sigmoid Activation
   Compute prediction probability for binary classification
   prob = sigmoid(logits)
   → Output: prob ∈ [0,1]

7. Prediction
   Compare prob against threshold to determine final label
   if prob >= threshold → toxic (1)
   else                 → non-toxic (0)
   → Output: predicted_label
```

Algoritma 2. *XLM-RoBERTa.*
End **Output:** prob, predicted_label

Tokenization dilakukan dengan menggunakan *tokenizer* bawaan XLM-RoBERTa untuk memastikan kesesuaian dengan *vocabulary* yang digunakan saat *pretraining* model. *Tokenization* bekerja melalui tiga tahap utama. Pertama, proses tokenisasi menggunakan metode *subword* yang memecah teks menjadi unit token, kemudian ditambahkan token spesial <s> sebagai penanda awal kalimat dan </s> sebagai penanda akhir kalimat. Kedua, token yang dihasilkan ditransformasikan menjadi *token IDs* untuk merepresentasikan setiap token dalam bentuk numerik sesuai dengan *vocabulary* model. Terakhir, dibentuk *attention mask* untuk membedakan antara token asli dan token *padding,* sehingga model hanya memproses informasi yang relevan.

Hasil *tokenization* kemudian masuk ke *backbone* XLM-RoBERTa untuk dilakukan *embedding*. Proses *embedding* pada model transformer dilakukan dengan mengubah teks menjadi representasi vektor berdimensi 768 melalui kombinasi beberapa komponen *embedding.* Setiap token hasil tokenisasi terlebih dahulu diubah menjadi *token embedding* yang merepresentasikan makna dasar token tersebut, kemudian ditambahkan dengan *positional embedding* sebagai informasi urutan token dalam *sequence*. Gambaran lebih lanjut mengenai proses *embedding* dari *input representation* dapat dilihat pada Gambar 3.4.



Gambar 3. 4 *Input representation* (Devlin dkk., 2019)

Setiap *token* direpresentasikan melalui penjumlahan *token embedding* dan *positional embedding* sebelum diproses oleh *transformer encoder.*

Selanjutnya, *transformer encoder layers* mengubah *input representation* menjadi representasi kontekstual yang lebih kaya melalui beberapa lapisan pemrosesan bertingkat dengan memanfaatkan mekanisme *self-attention,* sehingga setiap token dapat memahami hubungan dengan token lain dalam satu sekuen. Proses ini dilakukan secara berulang pada beberapa *layer* dalam *backbone* XLM-RoBERTa sehingga model dapat menghasilkan representasi teks yang semakin informatif. Proses *fine-tuning* model juga dilakukan di bagian ini dengan melibatkan *encoder layers* yang sudah dilatih sebelumnya pada tahap *pretraining.* Gambaran mengenai proses *fine-tuning* dapat dilihat pada Algoritma 3.

Algoritma 3. *Two-stage fine-tuning.*

```
Input: Pretrained model M, training data D
Start

1. Load Pretrained Model
   Load pretrained model M

2. Stage 1: Partial Fine-Tuning
   a. Freeze all encoder layers except the top 2 layers and
      classification head
   b. Train model for N1 epochs using re-weighting loss
   for epoch = 1 to N1:
    for each batch (x, y) in D:
```

Algoritma 3. *Two-stage fine-tuning.*

```
      i.   Perform forward propagation
           y_pred = M(x)

      ii.  Compute re-weighting loss
           loss = Re-weightingLoss(y_pred, y)

      iii. Update parameters of top 2 layers and
           classification head only

    → Output: partially fine-tuned model

3. Stage 2: Vanilla Fine-Tuning
   a. Unfreeze all model layers
   b. Train entire model layers for N2 epochs using BCE
    for epoch = 1 to N2:
    for each batch (x, y) in D:

     i.   Perform forward propagation
          y_pred = M(x)

     ii.  Compute BCE
          loss = BCE(y_pred, y)

     iii. Update all model parameters

   → Output: fully fine-tuned model

4. Return Final Model
   Return trained model M

End
Output: Trained model M
```

Strategi *fine-tuning* pada penelitian ini mengadopsi strategi dari penelitian (Valizadehaslani dkk., 2022) dengan tujuan untuk mengatasi *class imbalance* yang di mana *fine-tuning* dilakukan dengan dua tahapan utama. Tahap satu diterapkan untuk membentuk representasi awal yang seimbang antar kelas dengan *reweighting loss,* sehingga pada tahap dua *vanilla fine-tuning,* model tidak memulai dari kondisi bias terhadap kelas mayoritas, melainkan dari representasi yang sudah adil bagi kelas minoritas.

Berdasarkan penelitian (Valizadehaslani dkk., 2022), *fine-tuning* pada data *imbalance* sebaiknya diawali dengan melatih sebagian kecil layer *transformer* menggunakan *reweighting loss,* karena efek *re-weighting* paling terasa terjadi pada awal pelatihan dan *full fine-tuning* dengan *re-weighting* sejak awal berisiko mendistorsi *pretrained features.* Oleh karena itu, pada *stage 1* penelitian ini hanya dua *layer* transformer teratas dan *classification head* saja yang dilatih menggunakan *reweighting loss* untuk membentuk representasi awal *minority class* tanpa mengubah representasi bahasa umum hasil *pretraining.* Selanjutnya, *stage 2* melakukan *full fine-tuning* pada *embedding layer,* seluruh *encoder layer* dan *classification layer* menggunakan BCE untuk mengadaptasi representasi model secara global dan meningkatkan performa generalisasi model terhadap data baru.

Fase *warm-up* menggunakan BCE diterapkan secara konsisten pada seluruh konfigurasi eksperimen *fusion loss* sebelum komponen *fusion loss* diaktifkan, mengikuti prinsip (Cao dkk., 2019) yang menunjukkan bahwa model perlu membangun representasi awal yang stabil terlebih dahulu sebelum *loss* yang lebih kompleks diperkenalkan guna menghindari komplikasi optimasi awal pelatihan. Setelah fase *warm-up* selesai, *fusion reweighting loss* diaktifkan pada *stage 1* sejalan dengan temuan (Valizadehaslani dkk., 2022) bahwa efek *reweighting* paling efektif diterapkan pada tahap pelatihan awal sebelum seluruh *layer* model di optimasi secara penuh pada *stage 2.* Komputasi *loss* secara keseluruhan menerapkan Persamaan 1 sampai dengan Persamaan 7. Gambaran lebih lanjut mengenai proses komputasi *loss* pada model XLM-RoBERTa dapat dilihat pada Algoritma 4.

Algoritma 4. *Loss Computation*

Input: logits, targets, current_epoch, wu_epoch, fusion_type
Start

1. Compute individual losses
   loss_bce ← BCE(logits, targets)
   loss_fl ← FocalLoss(logits, targets)
   loss_drfl ← DRFL(logits, targets)

2. Warm-up phase (fusion loss only)
   if current_epoch < wu_epoch:
      return loss_bce

3. Fusion computation
   if fusion_type == "static_focal":
      loss ← $\omega_1 \cdot$loss_bce + $\omega_2 \cdot$loss_fl

   elif fusion_type == "static_drfl":
      loss ← $\omega_1 \cdot$loss_bce + $\omega_2 \cdot$loss_drfl

   elif fusion_type == "adaptive_focal":
      weights ← SoftAdapt(loss_bce, loss_fl)
      loss ← $\omega_1^{(t)} \cdot$loss_bce + $\omega_2^{(t)} \cdot$loss_fl

   elif fusion_type == "adaptive_drfl":
      weights ← SoftAdapt(loss_bce, loss_drfl)
      loss ← $\omega_1^{(t)} \cdot$loss_bce + $\omega_2^{(t)} \cdot$loss_drfl

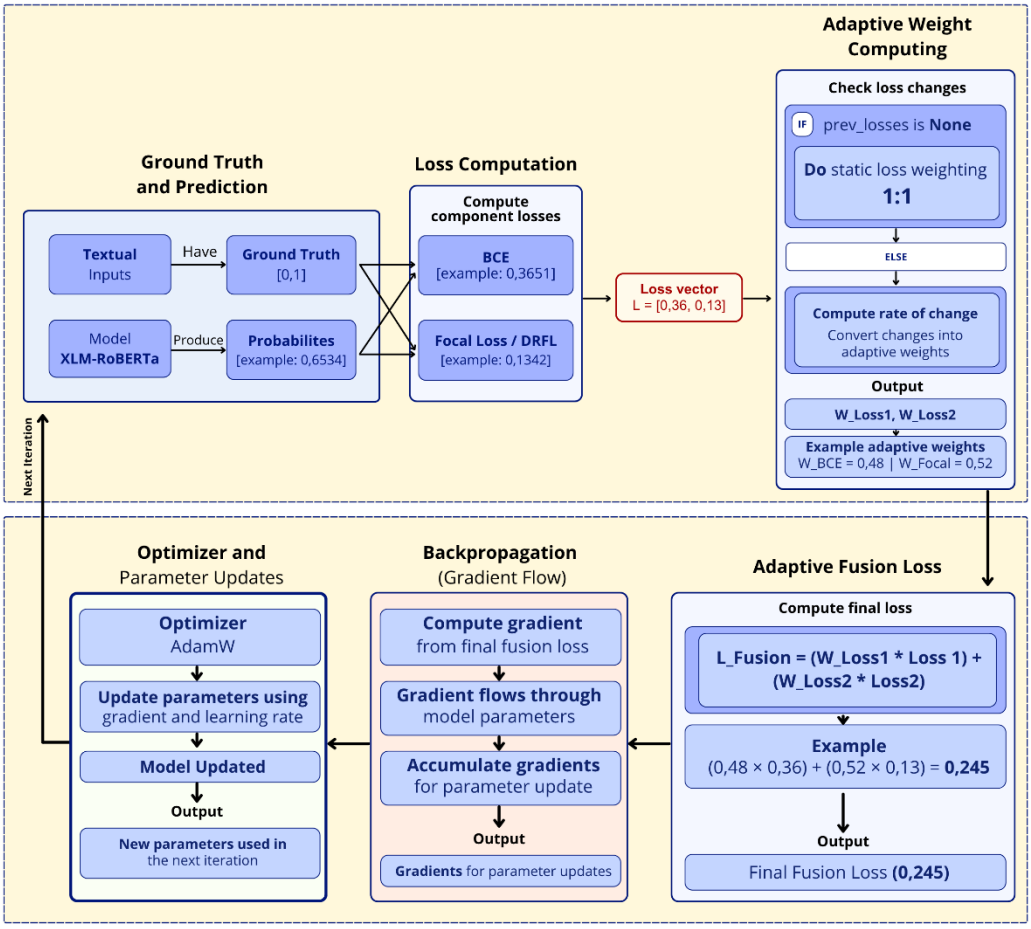
4. SoftAdapt weight computation (adaptive only)
   if prev_losses is None:
      weights ← [0.5, 0.5]
   else:
      s ← clamp((losses - prev_losses) / prev_losses, -10, 10)
      weights ← softmax(β · (s - max(s)))
   prev_losses ← losses

5. Return loss

End
Output: loss

Seluruh perhitungan individual komponen *loss* yaitu BCE, Focal Loss, dan DRFL dijadikan sebuah variabel untuk dipanggil pada proses selanjutnya. Pada fase *warm-up*, seluruh konfigurasi *fusion loss* mengembalikan BCE saja hingga *epoch* ke *wu_epoch* tercapai. Setelah *warm-up* selesai, *loss* digabungkan sesuai konfigurasi eksperimen, baik secara statis dengan bobot tetap α maupun secara

adaptif menggunakan mekanisme SoftAdapt yang menyesuaikan bobot w_1 dan w_2 secara dinamis berdasarkan laju perubahan *loss* antar *epoch*. Gambaran lebih lanjut mengenai ilustrasi proses *update model* menggunakan *Adaptive Fusion Loss* via *SoftAdapt* dapat dilihat pada Gambar 3.5.



Gambar 3. 5 Ilustrasi *Adaptive Fusion Loss*

Proses dimulai pada tahap *Ground Truth and Prediction*, yaitu model menghasilkan probabilitas prediksi yang dibandingkan dengan label sebenarnya untuk menghitung masing-masing komponen *loss*. Nilai *loss* kemudian dibentuk menjadi *loss vector* dan digunakan pada tahap *Adaptive Weight Computing*. Pada awal pelatihan, ketika nilai *loss* sebelumnya belum tersedia, kedua loss

menggunakan bobot statis 1:1, sedangkan pada iterasi berikutnya bobot dihitung secara adaptif berdasarkan laju perubahan masing-masing *loss*. Bobot yang dihasilkan selanjutnya digunakan untuk menghitung *Adaptive Fusion Loss* sebagai nilai *loss* akhir. Nilai tersebut kemudian digunakan pada proses *backpropagation* untuk menghitung dan mengakumulasikan gradien, yang selanjutnya digunakan oleh optimizer AdamW untuk memperbarui parameter model. Panah dari tahap *Optimizer and Parameter Updates* yang kembali ke tahap *Ground Truth and Prediction* menunjukkan bahwa parameter model yang telah diperbarui digunakan pada iterasi selanjutnya, sehingga proses dimulai kembali dengan menghasilkan prediksi baru dan berlangsung secara berulang sampai *max epoch* terpenuhi atau *early stopping* terpacu.

Konfigurasi *hyperparameter* pada penelitian ini sebagian besar mengacu pada penelitian sebelumnya (Song, Huang, & Xiao, 2021), meliputi *max length, batch size, optimizer,* dan *learning rate,* dengan penyesuaian pada *early stopping* dan *learning rate* untuk mengakomodasi skenario *two-stage fine-tuning* yang digunakan.

Proses *fine-tuning* menggunakan *learning rate* yang lebih kecil pada *backbone transformer* untuk menjaga stabilitas pelatihan serta mencegah perubahan parameter yang terlalu besar pada model yang telah melalui tahap *pre-training* (Devlin dkk., 2019). Selain itu, penelitian ini menerapkan aspek *differential learning rate,* yaitu perbedaan nilai *learning rate* antara *transformer encoder* dan *classification layer. Encoder* menggunakan *learning rate* lebih kecil karena sudah melalui *pretraining,* sehingga pembaruan bobot dilakukan secara hati-

hati. Sebaliknya, *classification layer* menggunakan *learning rate* lebih besar karena dilatih dari awal, sehingga memerlukan pembaruan yang lebih cepat untuk beradaptasi dengan tugas klasifikasi (Howard & Ruder, 2018). Pada *fine-tuning* tahap kedua bahkan ditetapkan *learning rate* yang lebih kecil lagi karena seluruh layer *pretrained* ikut diperbarui, sehingga diperlukan proses pembelajaran yang lebih hati-hati untuk meminimalkan risiko *catastrophic forgetting*.

Classifier head pada penelitian ini diadopsi dari arsitektur yang digunakan pada penelitian sebelumnya (Song, Huang, & Xiao, 2021), dengan menerapkan dua *fully connected layer* dan satu layer *output* sebagai lapisan klasifikasi di atas *backbone pretrained model*.

Terakhir, model dievaluasi menggunakan metrik akurasi, presisi, *recall*, dan *f1-score* dengan fokus pada F1 kelas minoritas untuk mengukur kemampuan model dalam mendeteksi kelas minoritas secara seimbang serta menghindari bias terhadap kelas mayoritas pada dataset *imbalanced*. *Threshold tuning* dilakukan pada tahap inferensi untuk mencari nilai batas keputusan klasifikasi agar performa model, terutama *f1-score*, lebih optimal pada dataset yang digunakan. Gambaran lebih lanjut mengenai proses *threshold tuning* dapat dilihat pada Algoritma 5.

Algoritma 5. *Threshold tuning*

```
Input: prediction probabilities (preds), true labels (golds),
threshold candidates
Start

1. Initialize
   best_f1 ← 0
   best_threshold ← 0.5

2. Threshold Evaluation
   For each threshold t in threshold candidates:

   a. Convert probabilities into binary predictions
      temp_preds ← (preds ≥ t)
```

Algoritma 5. *Threshold tuning*

```
   b. Compute minority-class F1-score
      f1 ← F1Score(golds, temp_preds, pos_label=1)

   c. Update best threshold
      if f1 > best_f1:
         best_f1 ← f1
         best_threshold ← t

3. Final Prediction
   final_preds ← (preds ≥ best_threshold)

End
Output: best_threshold, final_preds
```

Threshold tuning tidak diterapkan pada *stage 1* agar perbedaan performa antar konfigurasi *loss function* dapat diatribusikan secara langsung pada perbedaan *loss function*, bukan pada perbedaan *threshold* keputusan, sehingga seluruh eksperimen *stage 1* menggunakan *threshold default* 0.5. *Threshold tuning* hanya diterapkan pada model *final stage 2* dengan mencari nilai optimal berbasis *minority class f1-score* pada *validation set*, kemudian diterapkan secara *fixed* pada *test set* tanpa modifikasi, memastikan *test set* tetap sepenuhnya *unseen* selama proses optimasi sehingga hasil evaluasi mencerminkan kemampuan generalisasi model yang sesungguhnya.

Evaluasi model pada penelitian ini menggunakan data uji yang bersifat OOD, yakni data yang memiliki distribusi berbeda dari data latih. Mengacu pada (Yu dkk., 2024) distribusi *shift* dapat terjadi pada tingkat *covariate shift* maupun *concept shift*, dan evaluasi OOD lebih mencerminkan kondisi *deployment model* di dunia nyata di mana distribusi *shift* bersifat *ubiquitous* dan tidak dapat diprediksi sebelumnya. Data uji pertama, Jigsaw Toxic Comment, bahasa Inggris mengandung *covariate shift* ringan akibat perbedaan sumber pengumpulan data meskipun ruang label identik dengan data latih. Data uji kedua Ahmadizzan Toxic Dataset, bahasa

Indonesia mengandung *shift* yang lebih jauh, mencakup perbedaan sumber teks sekaligus perbedaan skema label yang mengindikasikan adanya *concept shift*, sehingga menguji kemampuan generalisasi dan adaptasi model secara lebih menyeluruh.

Selain dilakukan studi komparasi, studi ablasi pada *framework algorithm-level imbalance data handling* juga dilakukan pada penelitian ini untuk mengetahui kontribusi individual dari setiap komponen yang diterapkan dan seberapa besar kontribusi dari komponen-komponen tersebut pada *final model* jika seluruhnya digabung dan dikombinasikan. Beberapa konfigurasi yang diuji dapat dilihat pada Tabel 3.3.

Tabel 3. 3 Skenario studi ablasi

No	**Konfigurasi**	**Komponen yang aktif**
1	BCE	Tidak ada *imbalance handling*
2	Threshold Tuning	BCE dengan Threshold tuning
3	Static Fusion BCE with Focal Loss	Reweighting loss (statis)
4	Adaptive Fusion BCE with Focal Loss	Reweighting loss (adaptif)
5	Stage 2 (full fine-tuning)	Two-stage fine-tuning
6	Stage 2 dengan Threshold tuning	Threshold tuning

Studi ablasi dilakukan secara bertahap, dimulai dari model tanpa penerapan *imbalance handling* hingga model akhir yang mengombinasikan seluruh komponen metode yang diusulkan dalam satu konfigurasi. Skenario ini bertujuan untuk menganalisis kontribusi setiap komponen secara individual maupun kontribusi keseluruhannya terhadap peningkatan performa model pada kelas minoritas.