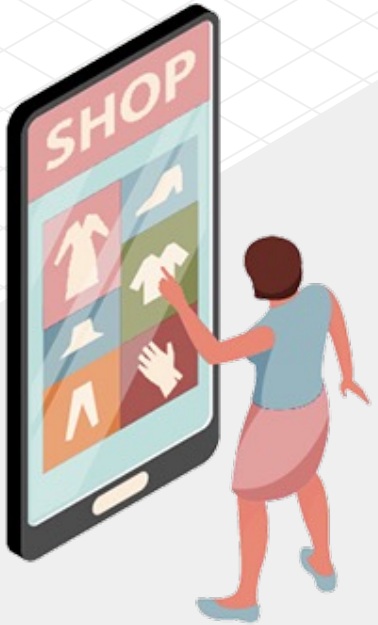

# Multi-Modal Duplicate Product Detection on Shopee

KM6312 BUBBLE TEAM

SHAN MINQIAN G2507390C
BAO CENCEN G2507353B
ZHANG JINGWEN G2507507D
QU YING G2506960G
SHEN YANBO G2503429B

# TABLE OF CONTENTS

# 01 Introduction



## Background

- Shopee's open marketplace allows multiple sellers to list the same product.
- This creates duplicate and near-duplicate listings with different titles, images, and seller info.

## Key Problems

- Cluttered search results filled with repeated items.
- Fragmented reviews/ratings, weakening social proof and trust.
- Higher user effort to identify truly distinct products.
- Lower search & recommendation efficiency, hurting CTR and conversions.

## Project Goal

- Build a multi-modal duplicate detection model using:
- Product titles, images, and structured attributes (price, brand, category).
- Automatically identify and group duplicate listings into unified product clusters.

## Business Value

- Cleaner search experience with fewer repeated items.
- Consolidated reviews → stronger trust and higher conversions.
- Fairer exposure for sellers by reducing duplicate spam.
- Supports platform growth through improved user experience and navigation.

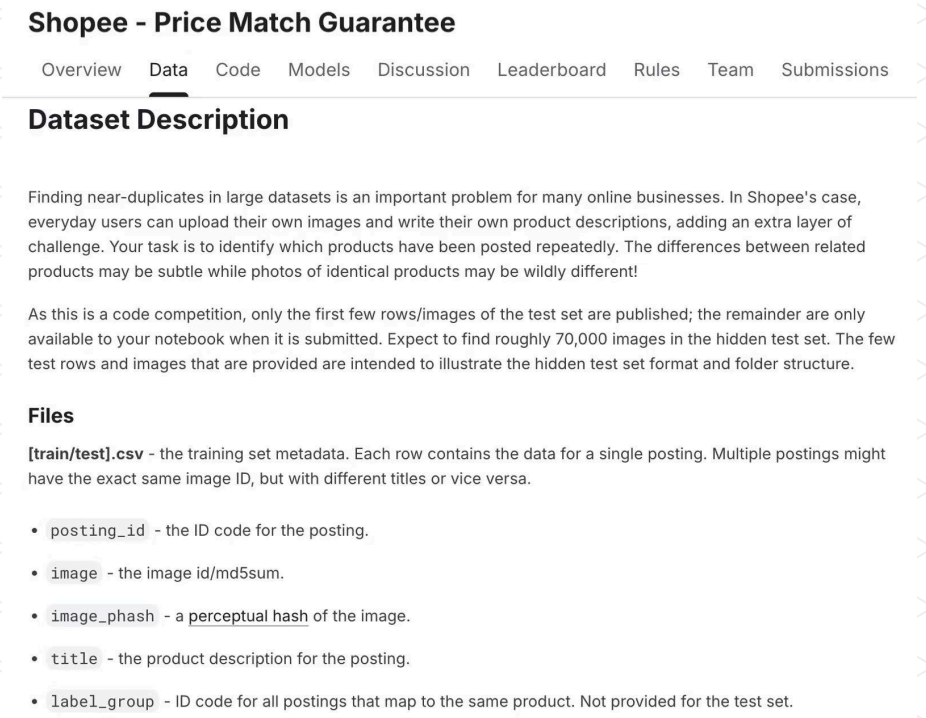
# 02 Data Preprocessing

# Dataset

- Official Shopee Product Matching (Kaggle)
- Product listings for duplicate-detection in e-commerce
- Used to train and evaluate multi-modal similarity models

## Key Fields

- posting_id — unique listing ID
- image — product image file
- title — seller-provided title
- label_group — ground-truth duplicate group ID



**Shopee - Price Match Guarantee**

Overview | Data | Code | Models | Discussion | Leaderboard | Rules | Team | Submissions

### Dataset Description

Finding near-duplicates in large datasets is an important problem for many online businesses. In Shopee's case, everyday users can upload their own images and write their own product descriptions, adding an extra layer of challenge. Your task is to identify which products have been posted repeatedly. The differences between related products may be subtle while photos of identical products may be wildly different!

As this is a code competition, only the first few rows/images of the test set are published; the remainder are only available to your notebook when it is submitted. Expect to find roughly 70,000 images in the hidden test set. The few test rows and images that are provided are intended to illustrate the hidden test set format and folder structure.

#### Files

**[train/test].csv** - the training set metadata. Each row contains the data for a single posting. Multiple postings might have the exact same image ID, but with different titles or vice versa.

- `posting_id` - the ID code for the posting.
- `image` - the image id/md5sum.
- `image_phash` - a perceptual hash of the image.
- `title` - the product description for the posting.
- `label_group` - ID code for all postings that map to the same product. Not provided for the test set.

## Text Cleaning (title_cleaned)

- Convert to lowercase
- Remove punctuation, emojis, HTML tags
- Remove repeated spaces
- Filter invalid or empty titles
- Output: title_cleaned

## Text Normalization (title_norm)

- Stopword removal
- Stemming / lemmatization
- Normalize number–unit formats (e.g., 128 GB ⟶ 128gb)
- Output: title_norm
- Used for TF-IDF, cosine similarity, and text embedding models.

```
import pandas as pd
import numpy as np
import re, random, unicodedata
import matplotlib.pyplot as plt
from pathlib import Path

RANDOM_SEED = 42
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)

DATA_DIR = Path('.')
TRAIN_PATH = DATA_DIR / 'train.csv'
FIG_DIR = DATA_DIR / 'figures'
OUT_DIR = DATA_DIR / 'artifacts'
FIG_DIR.mkdir(exist_ok=True)
OUT_DIR.mkdir(exist_ok=True)

df = pd.read_csv(TRAIN_PATH, low_memory=False)
print('Data shape:', df.shape)
display(df.head(3))

def clean_title(t):
    t = '' if pd.isna(t) else str(t)
    t = t.lower().strip()
    t = unicodedata.normalize('NFKC', t)
    t = re.sub(r'[^a-z0-9\s\-+]', ' ', t)
    t = re.sub(r'\s+', ' ', t)
    return t

df['title_clean'] = df['title'].apply(clean_title)
df[['title','title_clean']].head(5)
```

# Structured Feature Construction

**Brand Extraction**

- Rule-based matching using a predefined brand dictionary
- Examples: vivo, Xiaomi, Samsung, adidas
- Unrecognized brands → unknown

**Category Extraction**

- Keyword-based mapping to categories such as
- beauty
- home
- clothes
- phone
- stationery
- Low-frequency categories merged into Others

New fields created: brand, category

Limitation: Shopee titles mix languages and contain many niche brands → rule-based extraction has natural constraints.

```
Top brands after remap:
 unknown     30799
hp            1029
samsung        322
wardah         227
oppo           215
xiaomi         200
3m             154
emina          127
vivo           101
scarlett        93
apple           89
msi             68
pixy            66
loreal          61
adidas          57
Name: brand, dtype: int64
```

```
Category distribution (top 15):
 others         6574
beauty          5589
home            5464
clothes         5289
phone           2212
stationery      1669
bag             1648
food            1331
shoes            974
baby             966
electronics      899
auto             861
computer         621
sports           153
Name: category, dtype: int64
```

# Price Construction & Standardization

Shopee's raw dataset does not provide price, so we constructed it.

**Simulated Price (price)**

- Generated within reasonable e-commerce price ranges
- Adjusted loosely based on product type

**Standardized Price (price_std)**

- Z-score standardization
- Ensures consistent scaling for modeling
- New fields: price, price_std

```
Price summary (overall):
count    34250.00
mean        83.66
std        143.38
min          1.09
50%         38.98
90%        186.81
95%        327.43
max       2334.98
Name: price, dtype: float64
```

Sample rows:

| | category | price | price_std |
|---|---|---|---|
| **0** | bag | 60.030529 | -0.164814 |
| **1** | stationery | 8.037031 | -0.527452 |
| **2** | food | 15.687368 | -0.474093 |
| **3** | clothes | 43.957122 | -0.276921 |
| **4** | food | 3.101742 | -0.561874 |
| **5** | home | 36.624555 | -0.328063 |
| **6** | clothes | 26.993067 | -0.395240 |
| **7** | clothes | 20.679239 | -0.439277 |
| **8** | home | 79.197596 | -0.031131 |
| **9** | home | 47.951989 | -0.249058 |

# Exploratory Data Analysis (EDA)

To better understand the structure and distribution of the processed dataset, we conducted three key visual analyses:

- Category distribution
- Price distribution (log scale)
- Label group size distribution

These analyses help identify imbalance, noise patterns, and structural characteristics of the dataset.

# Exploratory Data Analysis (EDA)

## Top 10 Product Categories

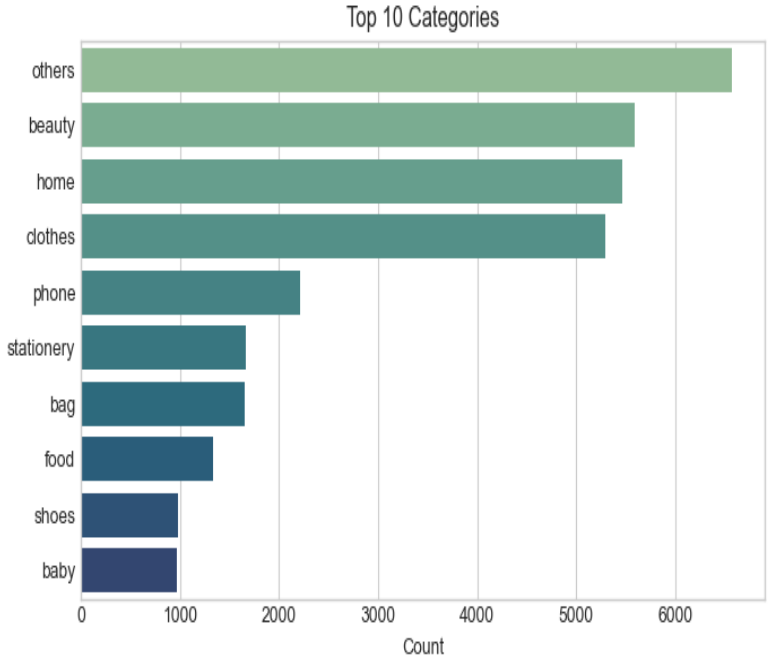
Interpretation:

•Others is the largest category due to rule-based extraction limitations & mixed-language titles.

•Major identifiable categories include beauty, home, clothes, phone, each with over 2,000–5,000 samples.

•Long-tail categories such as shoes and baby appear less frequently.

Indicates： imbalanced category distribution, which is common in e-commerce datasets.



# Exploratory Data Analysis (EDA)

## Price Distribution (Log Scale)

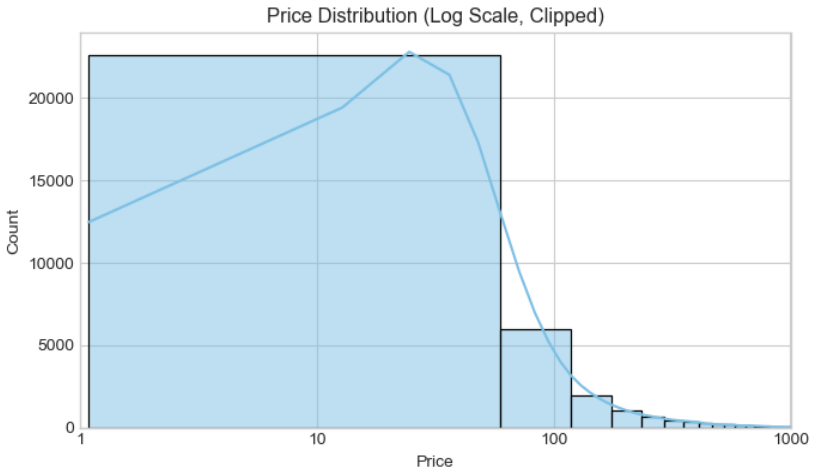
Interpretation:

•Simulated prices follow a realistic right-skewed pattern simila to marketplace price distributions.

•Most items fall in the low-to-mid price range (1–100 units).

•A few higher prices exist but are rare, matching typical long-ta price behavior.

Using log scale reveals the distribution clearly and reduces skewness.



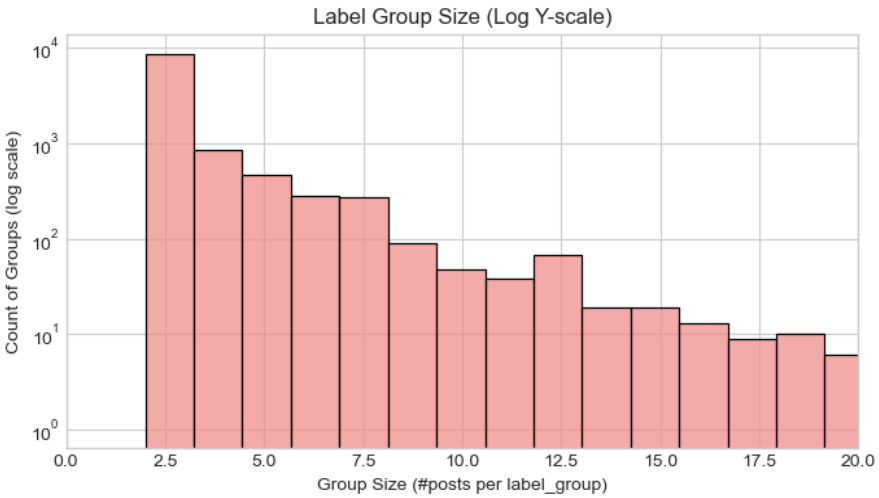
# Exploratory Data Analysis (EDA)

## Label Group Size (Duplicate Group Sizes)

Interpretation:

- Majority of label groups contain 2–3 items, meaning most duplicates are small clusters.
- A long-tail pattern exists, with a few groups containing 10–20+ items.
- Log-scale y-axis highlights the strong imbalance and sparsity of duplicate clusters.

This distribution confirms that duplicate detection is highly imbalanced, requiring careful sampling strategies.



# Image Preprocessing

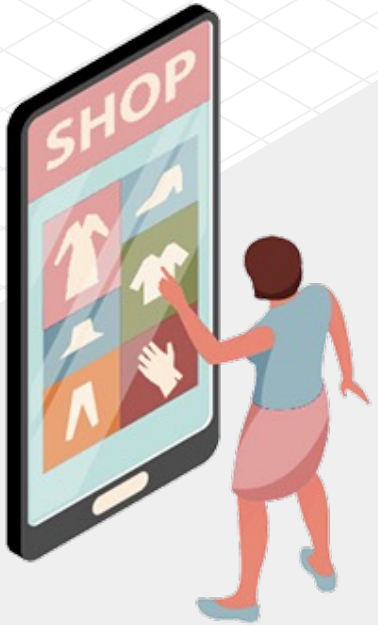
To ensure that all product images are consistent and usable, we applied simple but essential preprocessing steps:

- Converted all images to RGB format to unify color channels
- Resized images to a fixed resolution (e.g., 224×224) for consistency
- Removed unreadable or corrupted files to maintain data quality
- Extracted basic visual features, including
  - width and height
  - average RGB values
  - grayscale brightness

These steps make the images clean, standardized, and ready for multi-modal analysis.

# 03

# Text Modeling



# Text-only Duplicate Detection

**Objective:** Detect duplicate products using only textual data (title_clean)

**Approach:**

- Build **TF-IDF** baseline
- Apply **Sentence-BERT** for semantic matching
- Evaluate both using precision, recall, F1-score, accuracy

# Model Selection & Parameter Settings

**TF-IDF Baseline**

Vectorizer: TfidfVectorizer(max_features=5000)

Similarity: Cosine similarity

**Sentence-BERT**

Model: paraphrase-multilingual-MiniLM-L12-v2

Embedding batch size: 64

Similarity: Cosine similarity via util.cos_sim()

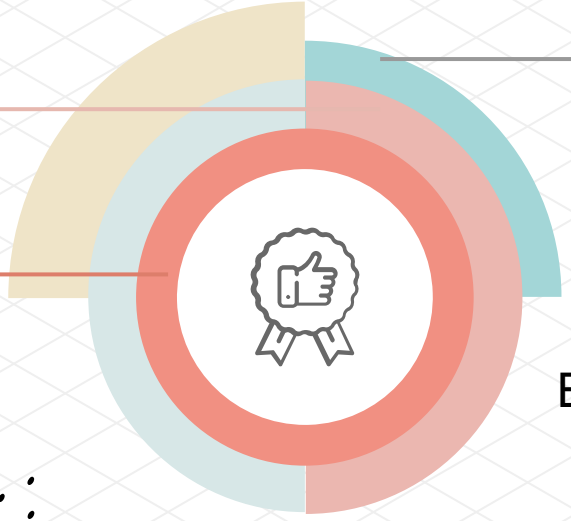
# Modeling Pipeline

**STEP 1**

Generate candidate pairs

**STEP 2**

Compute similarity scores (TF-IDF or SBERT)



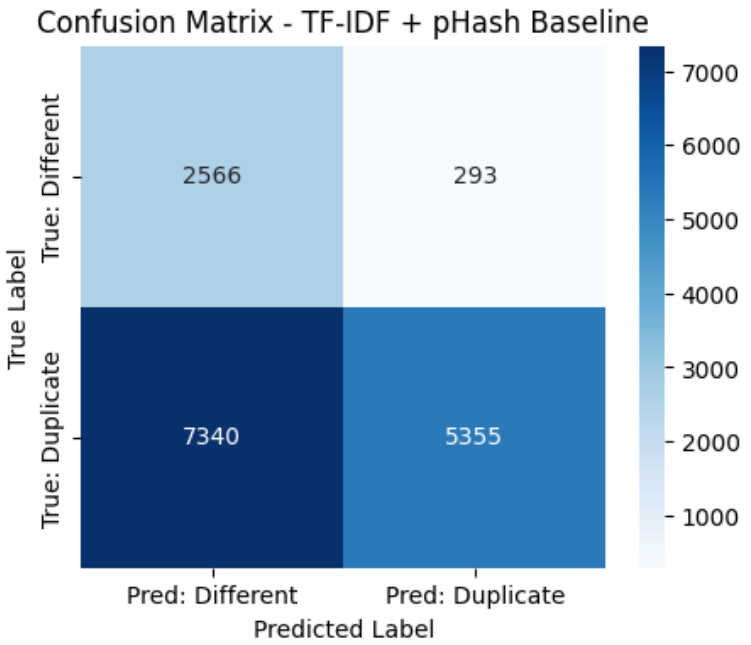
**STEP 3**

Predict duplicates using threshold = 0.7

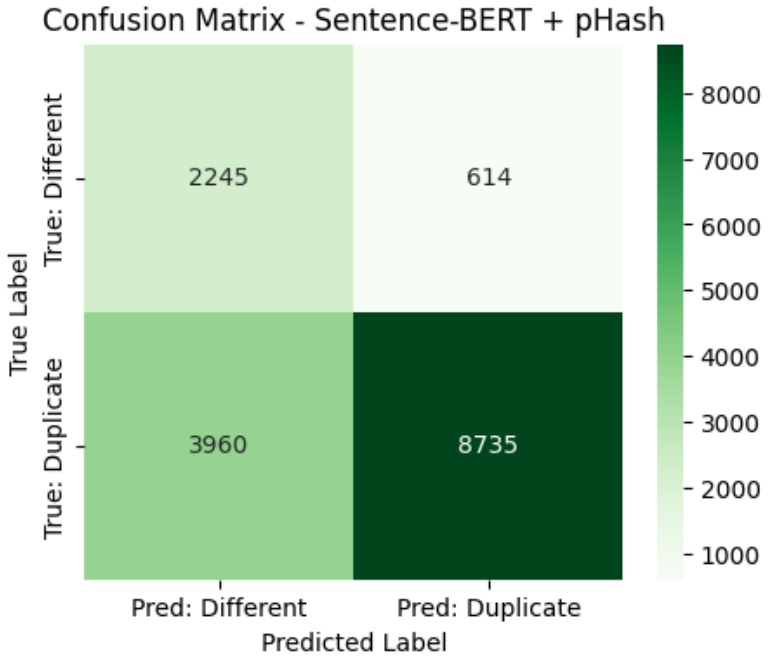
**STEP 4**

Evaluate against label_group

# Performance Comparison



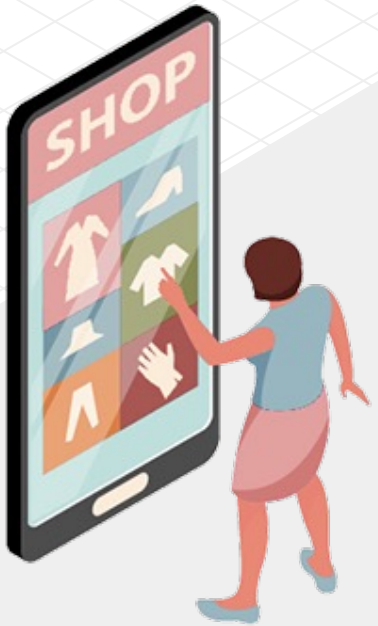

Accuracy: 0.5775 Precision: 0.9481
Recall: 0.4218 F1-Score: 0.5839




Accuracy: 0.7322 Precision: 0.9343
Recall: 0.6881 F1-Score: 0.7925

# 04

# Image Modeling



# EfficientNet B0

**EfficientNet**, a landmark architecture in automated deep learning, was conceived and developed by the Google Brain team (Tan, 2020). The architecture introduces a composite scaling method that can systematically extend the benchmark model. The extension method of EfficientNet has been verified in eight models (B0-B7). The method takes the compact B0 as the **benchmark model**, and then adopts composite expansion technology to gradually build a larger-scale and stronger model sequence from B1 to B7. In mobile or edge device application scenarios with limited computing resources, EfficientNet-B0 stands out for its excellent balance of efficiency and performance. Although larger-scale variants (B1-B7) have more parameters, **B0 can show strong accuracy in tasks such as image classification and target detection, making it an ideal choice for calculating budgets** (Kansal, 2024).

This study uses a **pre-trained EfficientNetB0 model** to extract high-level visual embeddings. The Shopee product dataset contains more than **34,000 product images**. EfficientNetB0 provides a strong accuracy–efficiency trade-off and can extract embeddings for 30K+ images within reasonable runtime on a laptop environment.

# Perceptual hashing (pHash)

Perceptual hashing is an advanced hash technology, which is originally designed to make the human perception system think that similar media content can produce the same or similar hash values.

Perceptual hashing's processing process include (Zauner, 2010):

- **Convert the image to grayscale first.**
- **Blurring and gamma correction**
- **Extract features and build hashes**

This project uses pHash to generate a 64-bit perceptual signature for each image, which can be used for **coarse similarity filtering** and eliminating obviously different pairs.

# Image-only Experiment
# EfficientNet B0

Image Preprocessing:

- Resize to 224×224 — required by CNN backbones
- Normalize pixel values to [0,1]
- Convert BGR → RGB (OpenCV default → TensorFlow format)
- Handle corrupted and missing images

Embedding Extraction:

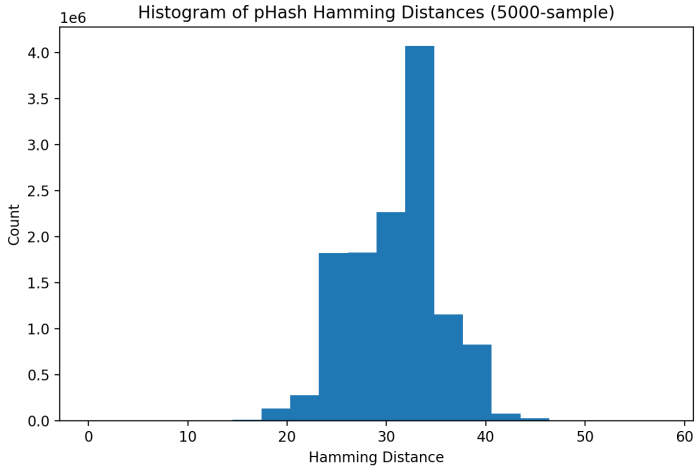
- Use EfficientNet-B0 without classification head
- Global Average Pooling → Each image → 1280-dimensional embedding vector

Output Example

- Shape: (34,250 images × 1280 features)

# Image-only Experiment
## pHash

- The histogram shows an obvious multi-peak structure:
- The **vast majority** of unrelated product pairs are in the range of **22-40**.
- Small clusters of medium-similarity goods are concentrated in the range of **10–18**.
- Only a few pairing distances are **less than 10**, which strongly suggests similar duplicate or highly similar goods.



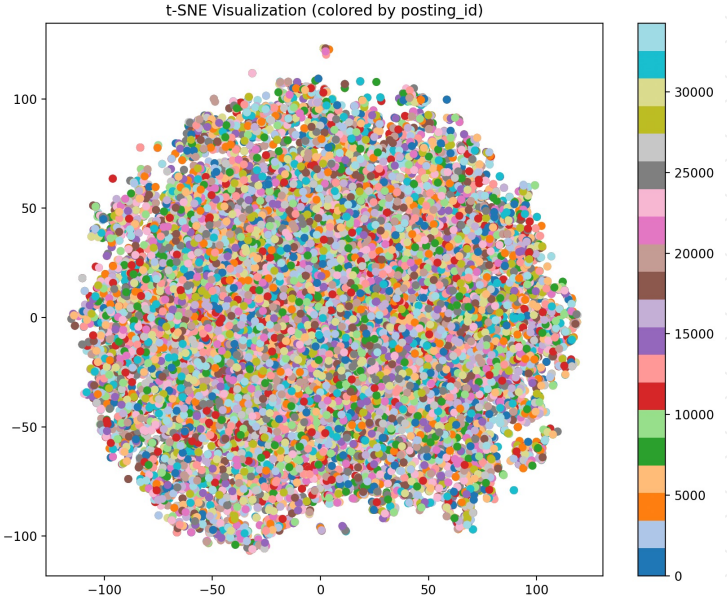

Based on this empirical distribution, we evaluate the candidate threshold from the conceptual level:

- Although the threshold below 10 can achieve extremely high accuracy, it will omit a large number of legal similar goods (low recall rate).
- A threshold higher than 18 will significantly expand the candidate pool, increase the calculation overhead and introduce excessive false alarms.
- Therefore, **selecting the threshold of Hanming distance ≤15** can achieve practical balance - not only retaining most visual similarity terms, but also effectively filtering irrelevant terms.

# Key Findings from Image-only

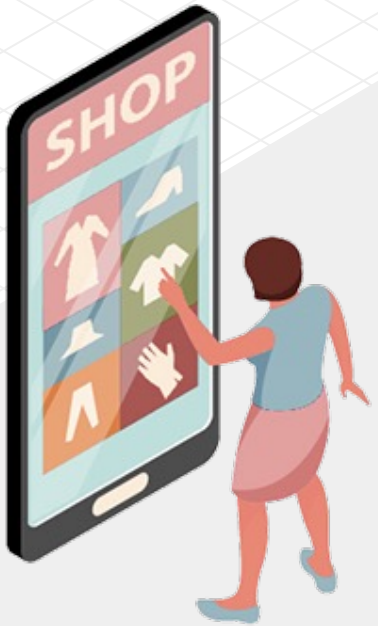
To present the structure of the 1280-dimensional embedding space more intuitively, we employed t-SNE to reduce it to a two-dimensional plane. Each point corresponds to a product image and is colored according to its posting_id. Since posting_id can uniquely identify each image, the color presents **a random mixed state**, which is in line with expectations.

Although global clusters cannot be observed when coloring according to posting_id, local neighborhoods can still **capture visual similarity**, this meets the requirements of repeated commodity detection.



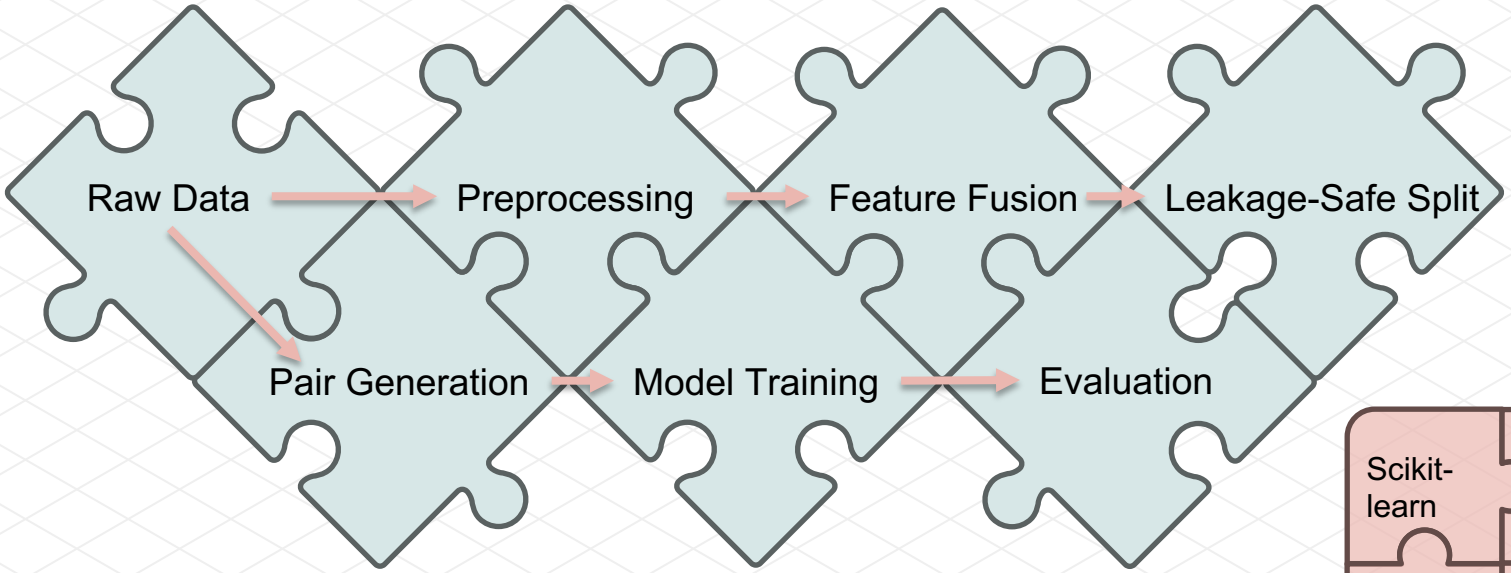
# 05

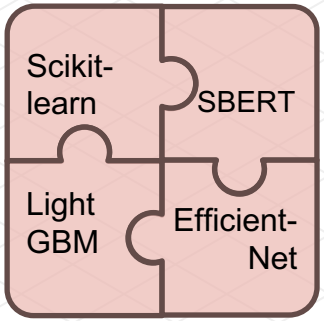
# Multi-Modal Fusion



# Multi-Modal Fusion

## Technical Architecture Overview



# Multi-Modal Fusion

## Technical Architecture Overview

- End-to-end pipeline: raw CSV ⟶ deployable model
- Modular design: preprocessing, fusion, training, evaluation
- Leakage-safe split by posting_id
- Core stack: Scikit-learn, LightGBM, SBERT, EfficientNet
- Automated and reproducible workflow

Python

```python
# Step 1: Load raw data
df = pd.read_csv("shopee_dataset_final.csv")

# Step 2–3: Preprocessing + Fusion
→ SBERT + EfficientNet + Structured features → 1676-dim vector

# Step 4–5: Split + Pair generation
→ Leakage-safe split → Balanced positive/negative pairs

# Step 6–10: Training pipeline
→ CV → Tuning → Final model → Evaluation

# Step 11–14: Output & Deployment
→ Visualizations → Feature importance → Business insights → Save model
```

# Multi-Modal Fusion

## Feature Engineering

### Feature Modalities

| Modality | Dimension | Implementation |
|---|---|---|
| **Text** | 384 | Sentence-BERT embeddings |
| **Image** | 1280 | EfficientNet embeddings |
| **Structured** | 12 | Price, brand, category, image statistics |

- Unified feature vector: 1676 dimensions

### Pairwise Feature Construction

Python

```python
# Core feature combinations
feature_diff = |X_i - X_j|           # Absolute difference
hadamard = X_i * X_j                 # Element-wise product
cosine_sim = dot(X_i_norm, X_j_norm) # Cosine similarity
same_brand = 1 if brand_i == brand_j # Brand flag
```

# Multi-Modal Fusion

## Model Design & Training

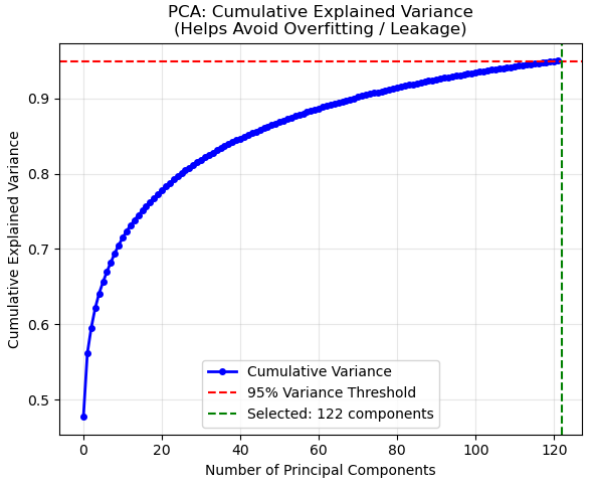
### Leakage Prevention

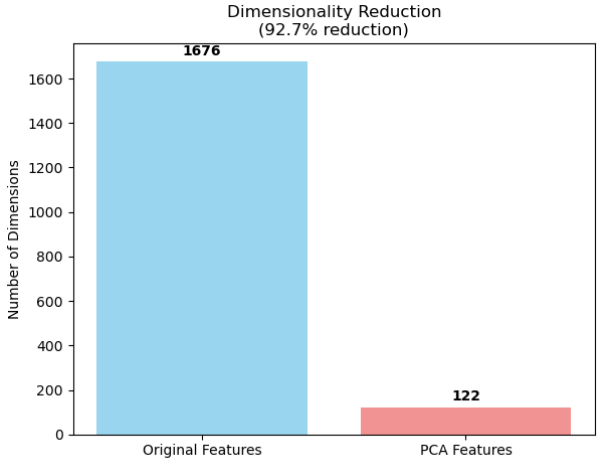
```python
# Key safeguards
1. Split data by posting_id
2. Fit scaler & PCA only on training set
3. Strict train/test isolation
```

- PCA fitted only on training set

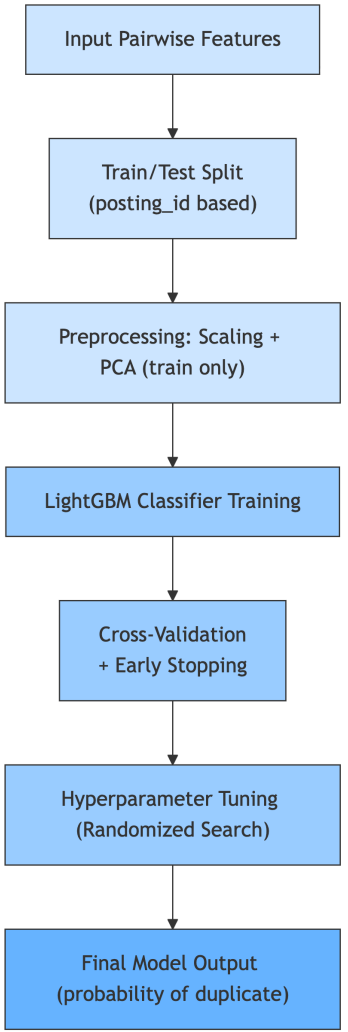
### PCA Cumulative Explained Variance Curve

- Dimensionality reduction: 1676 → 122 components (95% variance)



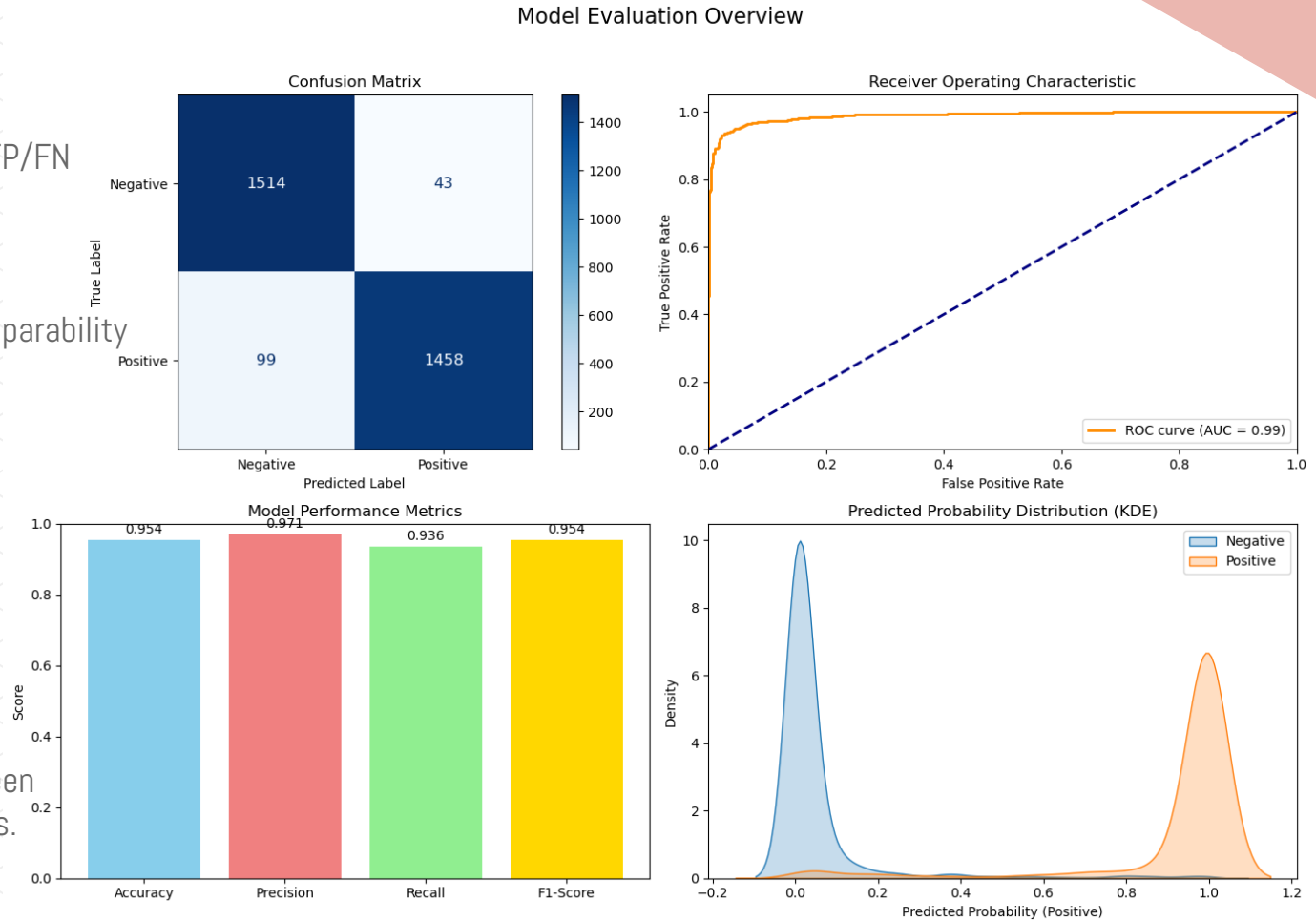

LightGBM Training Workflow



# Multi-Modal Fusion

## Performance Analysis

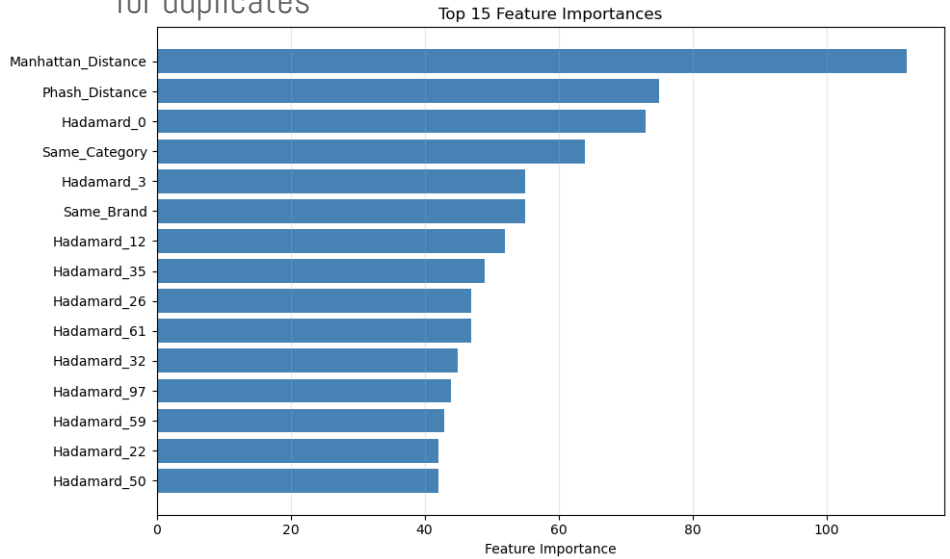
- Confusion matrix shows low FP/FN
- AUC: 0.9883
- ROC curve indicates strong separability
- Accuracy: 95.44%
- Precision: 97.14%
- Recall: 93.64%
- F1-Score: 95.36%
- KDE probability distribution shows clear separation between duplicates and non-duplicates.



# Multi-Modal Fusion

## Feature Importance Analysis

- Manhattan distance (text embeddings): strongest semantic signal for pairwise similarity
- pHash similarity (image): captures visual resemblance, useful for hard negatives
- Hadamard product (text): encodes interaction between embeddings
- Category/Brand match flags: provide strong structural cues for duplicates

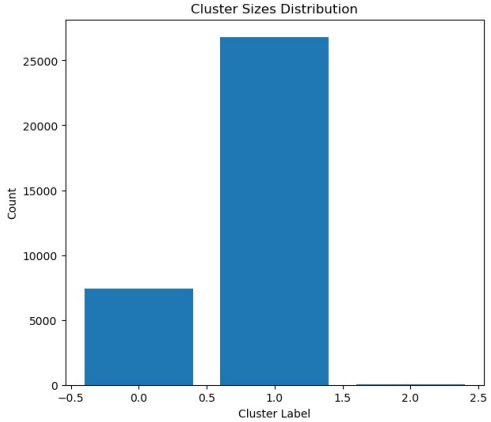


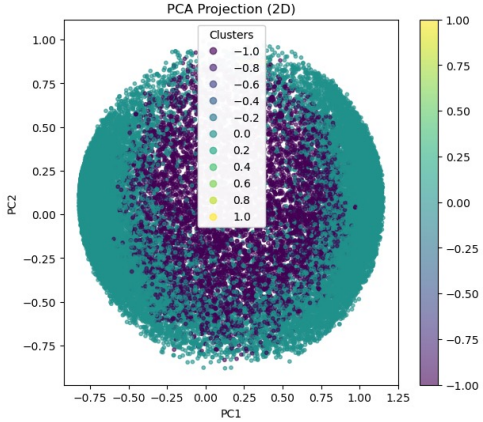

## Ablation Study & Insights

- Text + Numeric features yield best performance (F1 = 0.9873)
- Text only performs nearly as well (F1 = 0.9866)
- Text features dominate the overall signal
- Selective fusion is necessary

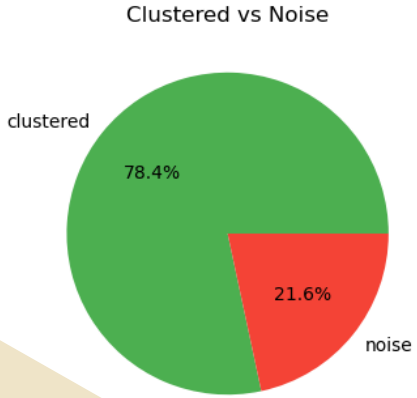
| Feature Set | F1 | AUC | Rank |
|---|---|---|---|
| Text + Numeric | 0.9673 | 0.9917 | 1 |
| Text Only | 0.9656 | 0.9924 | 2 |
| Text + Image | 0.9565 | 0.9883 | 3 |
| All Features | 0.9560 | 0.9885 | 4 |
| Image + Numeric | 0.8798 | 0.9396 | 5 |
| Image Only | 0.8792 | 0.9381 | 6 |
| Numeric Only | 0.8718 | 0.9341 | 7 |

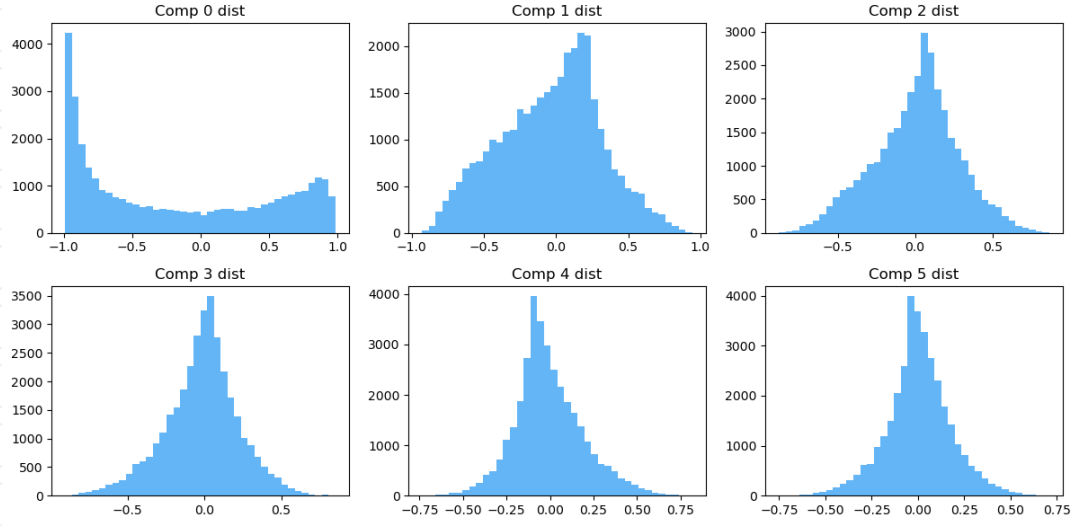
# Preliminary Clustering Attempt




- DBSCAN on PCA-reduced features (122 components)
- Suggested eps: 0.20 (cosine distance)
- Two major clusters identified
- PCA projection shows clear separation
- Clustered vs Noise: 78.4% vs 21.6%
- Component distributions mostly Gaussian-like
- Potential for unsupervised grouping



# 06
# Expected Outcomes & Business Value

# Commercial Problem & Motivation

- Shopee marketplace suffers from duplicate and near-duplicate product listings
- Consequences: search clutter, fragmented reviews, weak credibility, lower conversion
- Duplicate listings reduce user trust and increase bounce rate

Key metrics affected: CTR ↓, CVR ↓, GMV ↓

- Business need: automated, scalable duplicate detection → quality, fairness, revenue

# Expected Outcomes
## – Platform-Level Impact

| AREA | OUTCOME |
| --- | --- |
| Search Quality | Cleaner interface, higher content relevance |
| User Behavior | Faster search navigation, higher trust |
| Engagement Metrics | CTR ↑ due to more meaningful clicks |
| Purchase Confidence | CVR ↑ through unified ratings & reviews |
| Revenue Impact | GMV ↑ due to CTR × CVR uplift |

# How Duplicate Detection Drives Business Metrics

Cleaner Catalog → Better Guidance → Higher Trust → More Purchases

| Business Metric | Impact |
| --- | --- |
| CTR ↑ | Reduced noise → More relevant clicks |
| CVR ↑ | Centralized reviews → Stronger purchase decision |
| GMV ↑ | Higher CTR and CVR lead directly to revenue growth |
| Bounce Rate ↓ | Less frustration → Higher retention |

# Marketplace Ecosystem & Governance Impact

- Prevents sellers from posting duplicate listings to manipulate exposure
- Ensures fair visibility for authentic and small sellers
- Strengthens platform governance and policy enforcement
- Reduces customer disputes and misleading product claims
- Builds a trusted and sustainable marketplace ecosystem

# Operational and Strategic Benefits

| Area | Benefit |
| --- | --- |
| Moderation Cost | Automated duplicate detection reduces manual review burden |
| Efficiency | Approves quality listings faster, filters fake/repeated uploads |
| Content Quality | Standardizes catalog, improves metadata consistency |
| Scalability | Suitable for real-time deployment at platform level |

# Business Value Chain

Multimodal Duplicate Detection

↓

Cleaner Search Interface & Unified Product Pages

↓

CTR ↑  CVR ↑  Trust ↑

↓

GMV ↑  Seller Fairness ↑  Moderation Cost ↓

↓

Platform Governance & Long-Term Sustainability

- Shows how technical model links to business metrics & ecosystem health
- Demonstrates cascading value from search quality to marketplace trust
- Highlights both revenue and governance dimensions

# 07 Conclusion

# Final Takeaway: What This Project Proves

**Multimodal fusion (Text + Image + Metadata) outperforms single-source detection**

- pHash improves candidate retrieval speed and complements deep models
- Duplicate detection is not only a technical challenge, but a commercial enabler
- Direct impact on: search quality, CTR, CVR, GMV, fairness, and governance
- Supports real-world deployment in Shopee-like e-commerce environments

# Future Deployment Potential

Real-time duplicate detection at upload

Multilingual brand ontology expansion

Unified Product Page (UPP) for review consolidation

Business KPI dashboard (CTR, CVR, GMV tracking)

Integration into platform policy & fraud detection systems

# THANKS