Vietnamese-German University

COMPUTER SCIENCE ENGINEERING

Thesis Defense

Detecting Phishing Emails Using Deep Learning Models

By: Nguyen Pham Truong Nhat - 17422

TABLE OF CONTENT

I. Introduction

- Over 3.4 billion phishing emails are sent every day.
- Google blocks 100 million malicious emails daily.
- 48% of emails worldwide in 2022

II. Developed Phishing Email Detection System Architecture

1. Naive Bayes
2. Logistic Regress
3. Decision Tree
4. XGBoost
5. Random Forest

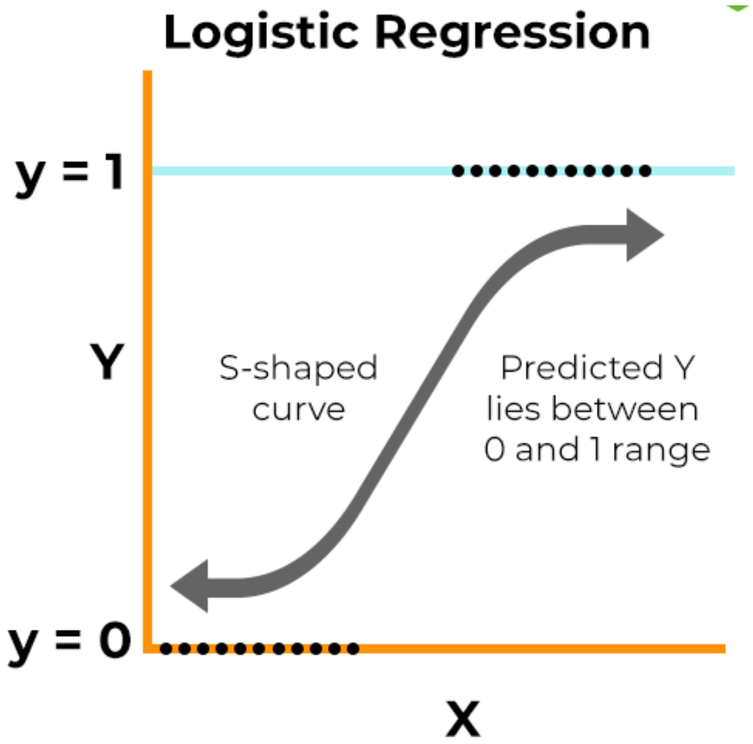
1. Naive Bayes

- Lightweight, highly efficient
- Both small and large datasets
- “Naive” has both benefits and drawbacks
- Zero Frequency

$$\mathbb{P}(S \mid x_1, \ldots, x_n) \approx \frac{\mathbb{P}(S) \prod_{i=1}^{n} \mathbb{P}(x_i \mid S)}{\mathbb{P}(S) \prod_{i=1}^{n} \mathbb{P}(x_i \mid S) + \mathbb{P}(H) \prod_{i=1}^{n} \mathbb{P}(x_i \mid H)}$$

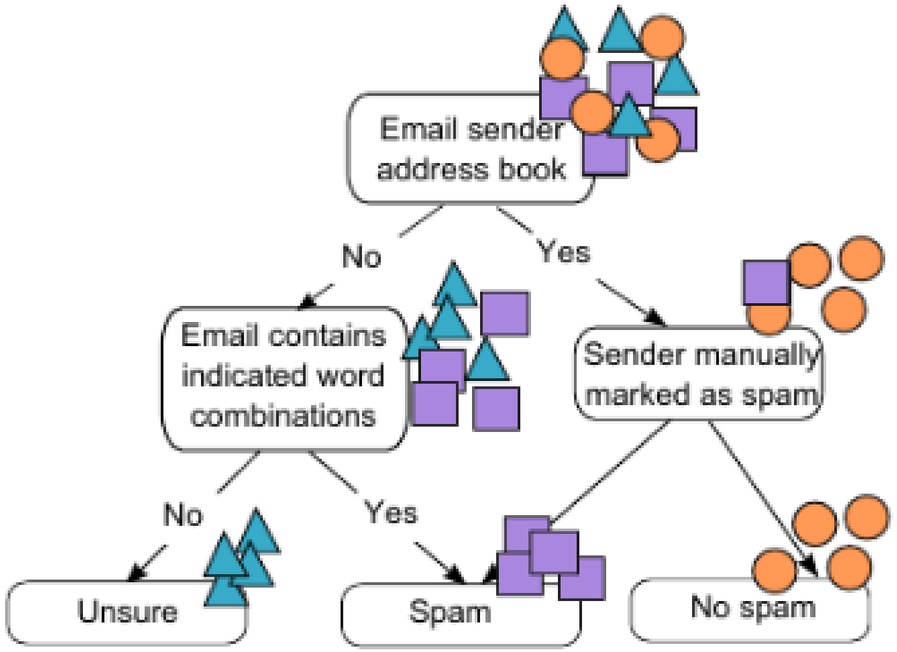
2. Logistic Regression

- Relationships between text contexts and class labels
- Lightweight, highly efficient
- Both small and large datasets
- Struggle with complex text patterns



3. Decision Tree

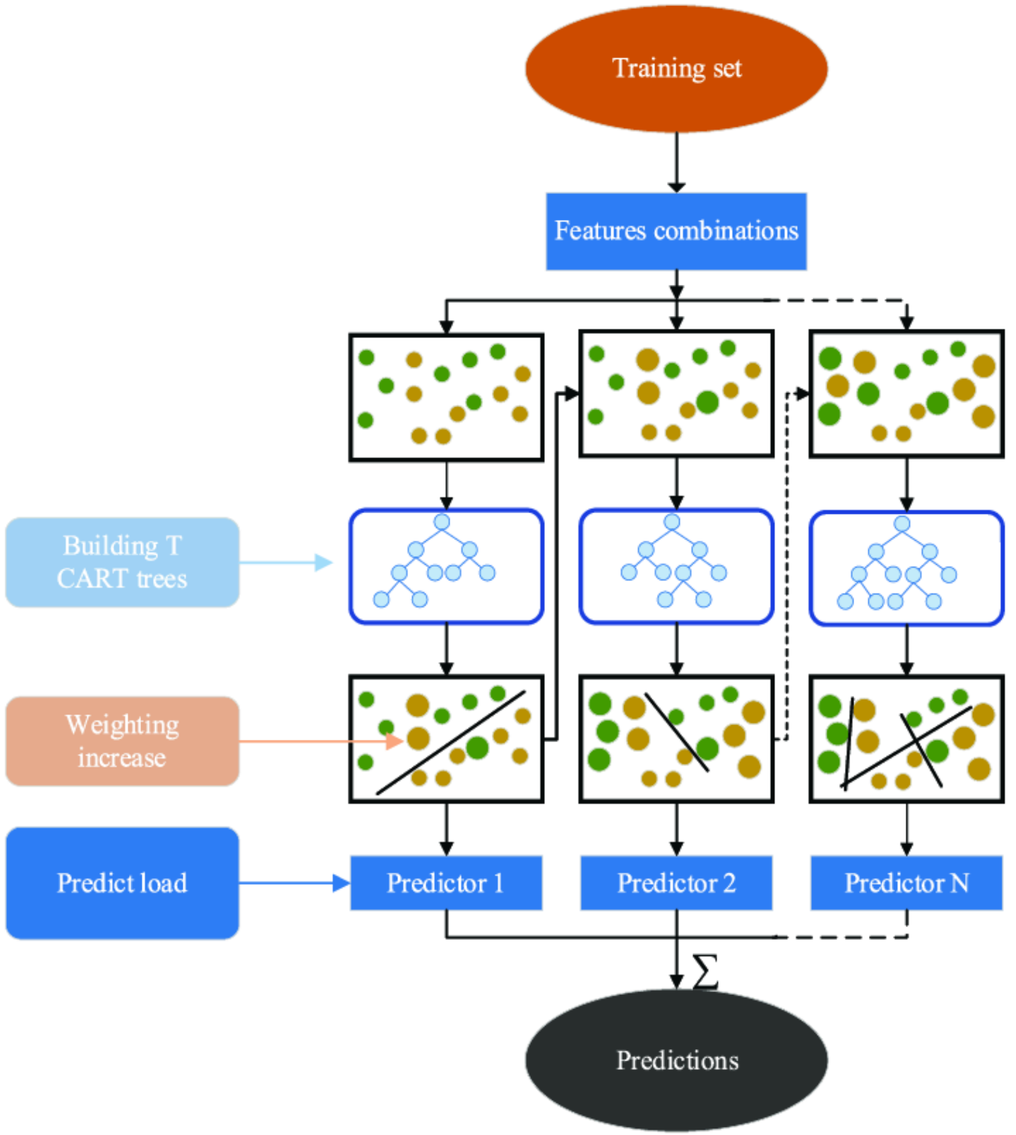
- Root Node
- Subsets (Internal Nodes)
- Feature Values (Leaf Nodes)
- Simplicity, interpretability



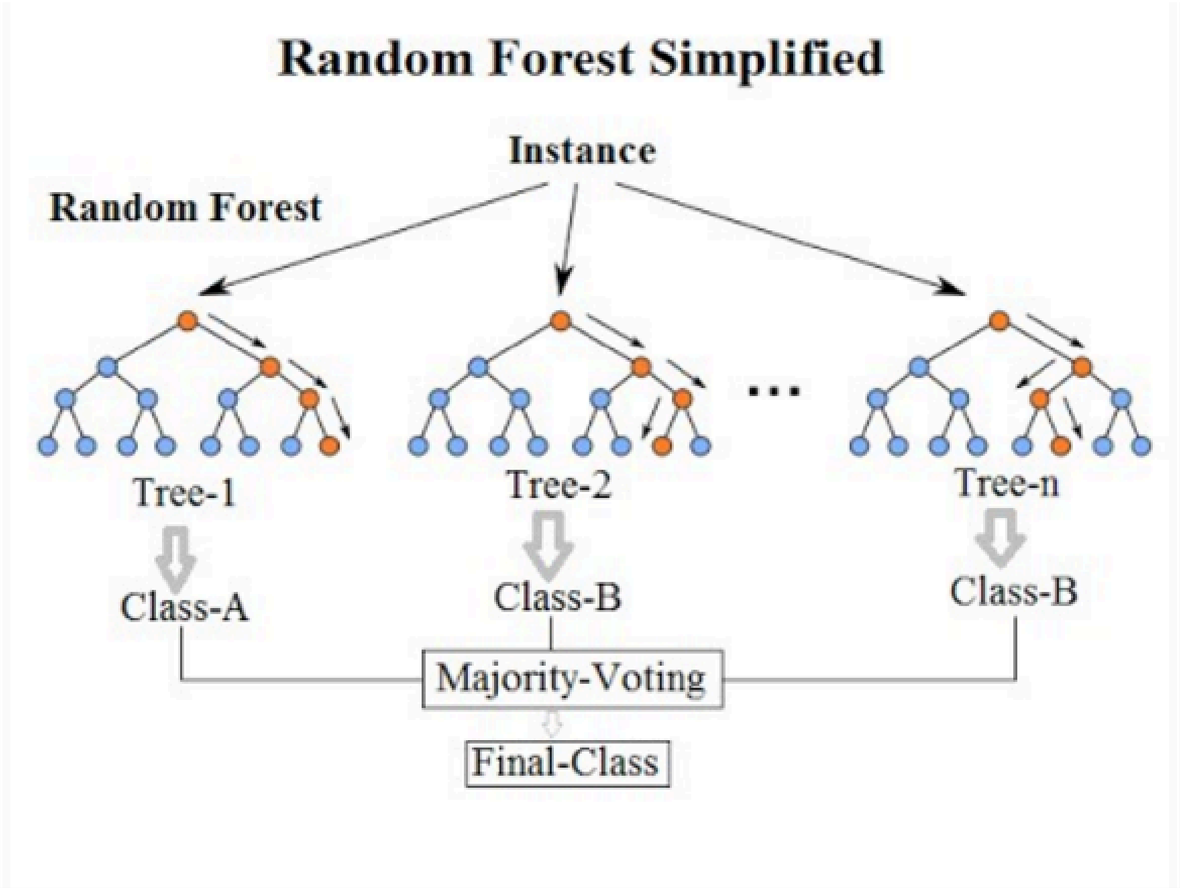
4. XGBoost

- Apply the TF-IDF technique
- Multiple Decision Trees (Weak Learners)
- High computational resources and time-consuming

4. XGBoost


Training set
Features combinations
Building T
CART trees
Weighting
increase
Predict load
Predictor 1
Predictor 2
Predictor N
Σ
Predictions

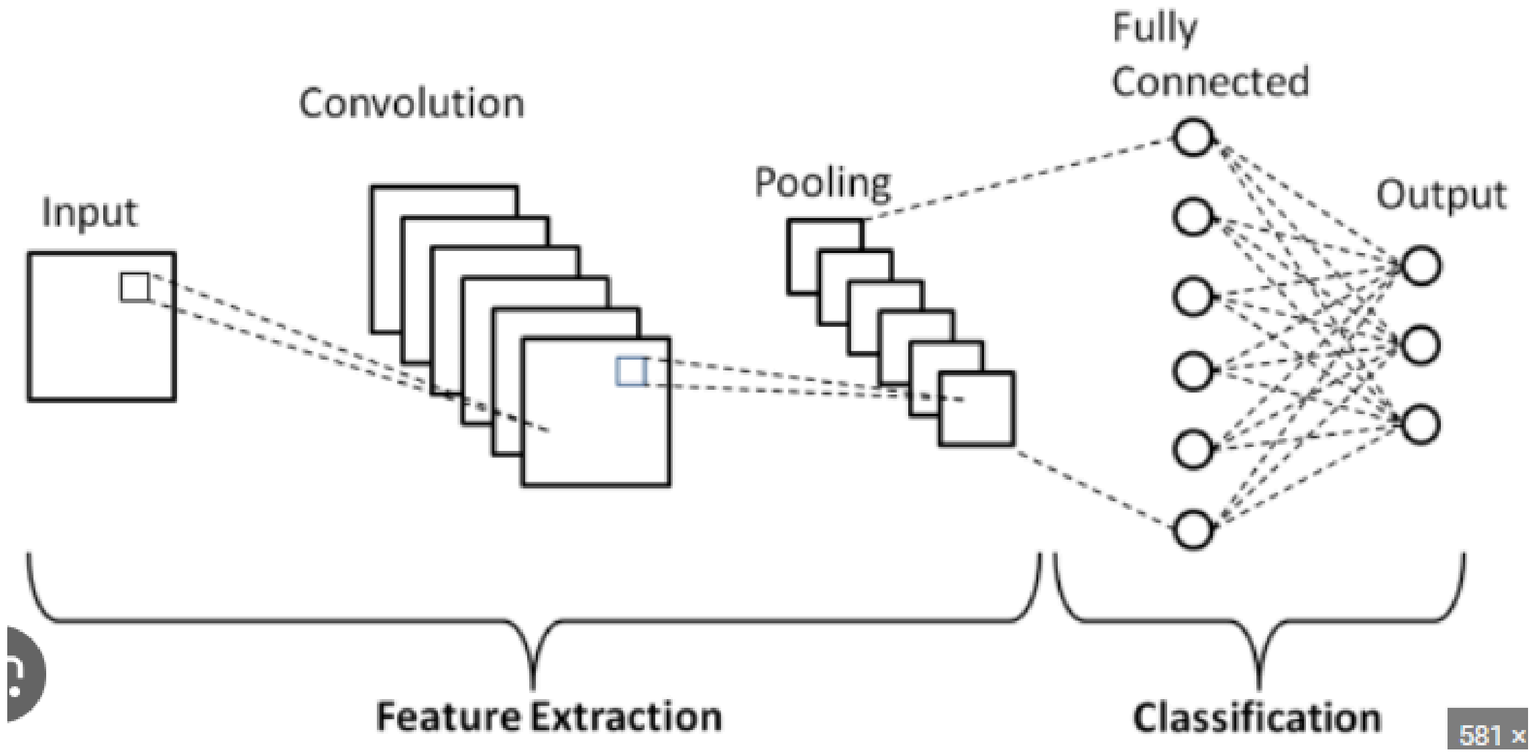
5. Random Forest



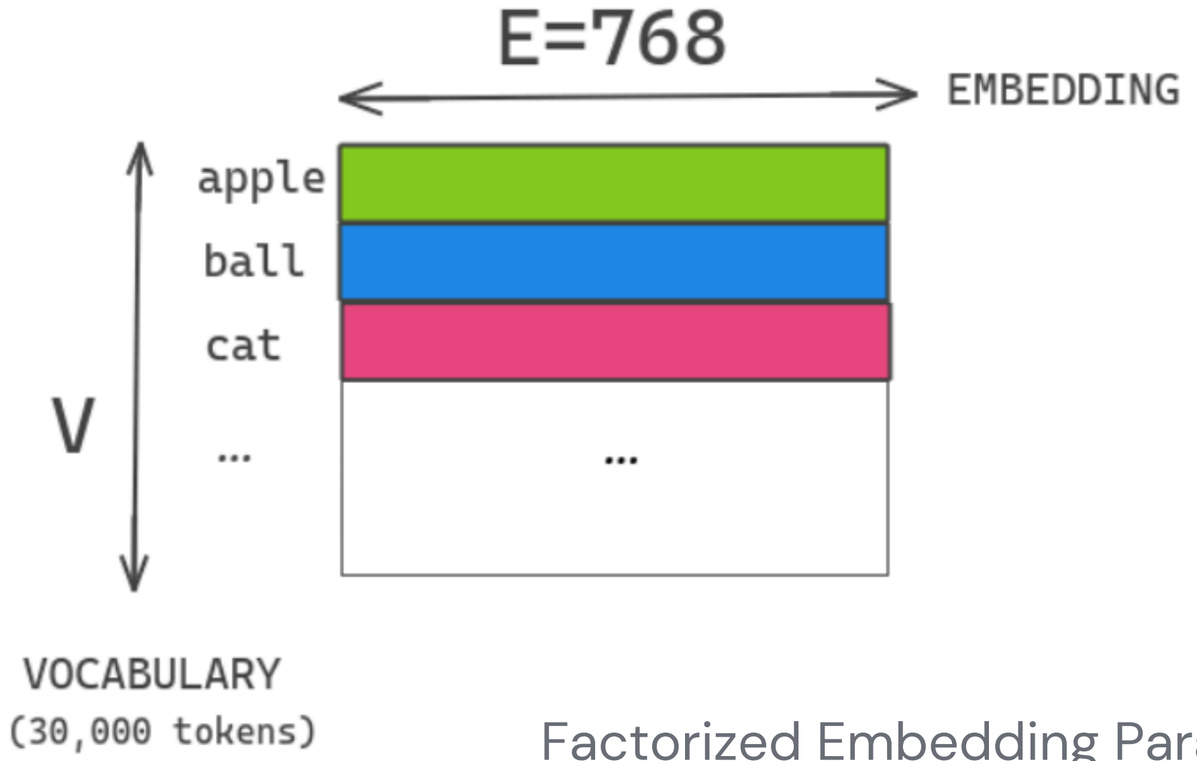
III. Proposed Phishing Email Detection System Architecture

1. Convolutional Neural Network
2. ALBERT
3. TinyBERT
4. ELECTRA

1. CNN


Fully
Connected
Convolution
Pooling
Output
Input
Feature Extraction
Classification
581 ×

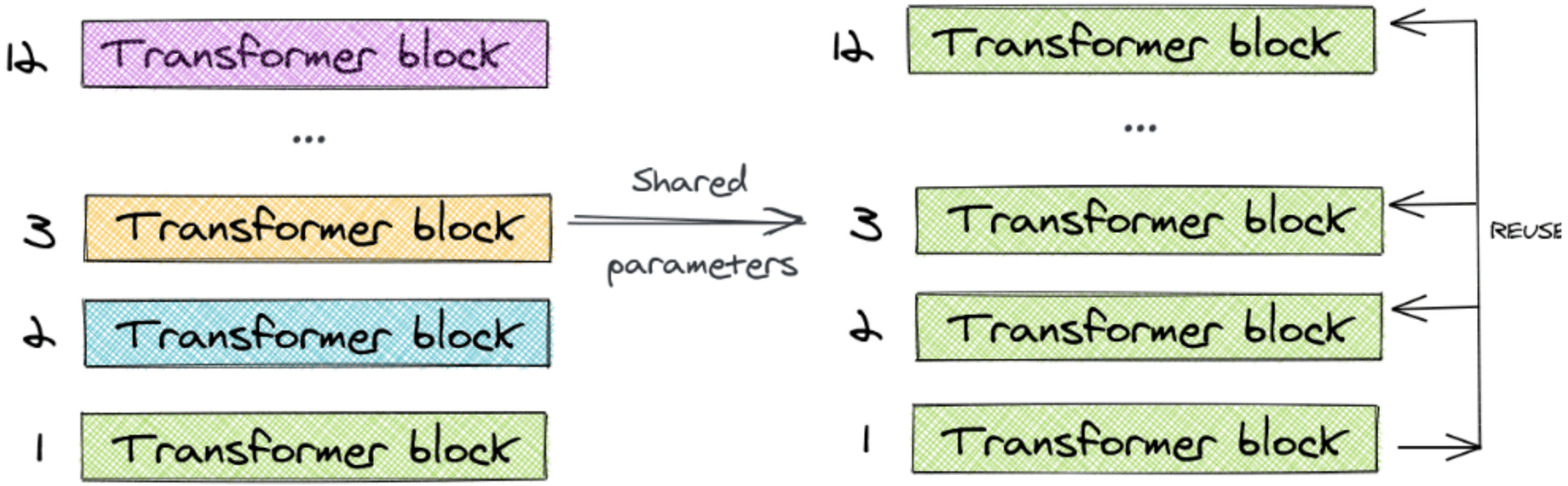
2. ALBERT



$$V \times E \rightarrow E \times H = V \times H$$

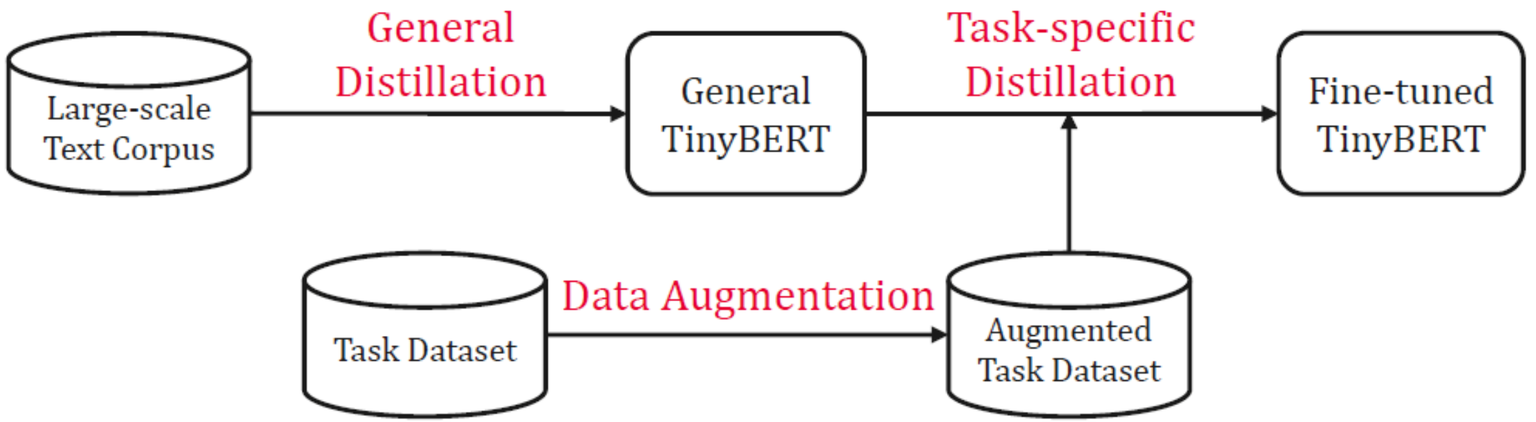
Factorized Embedding Parameterization

2. ALBERT

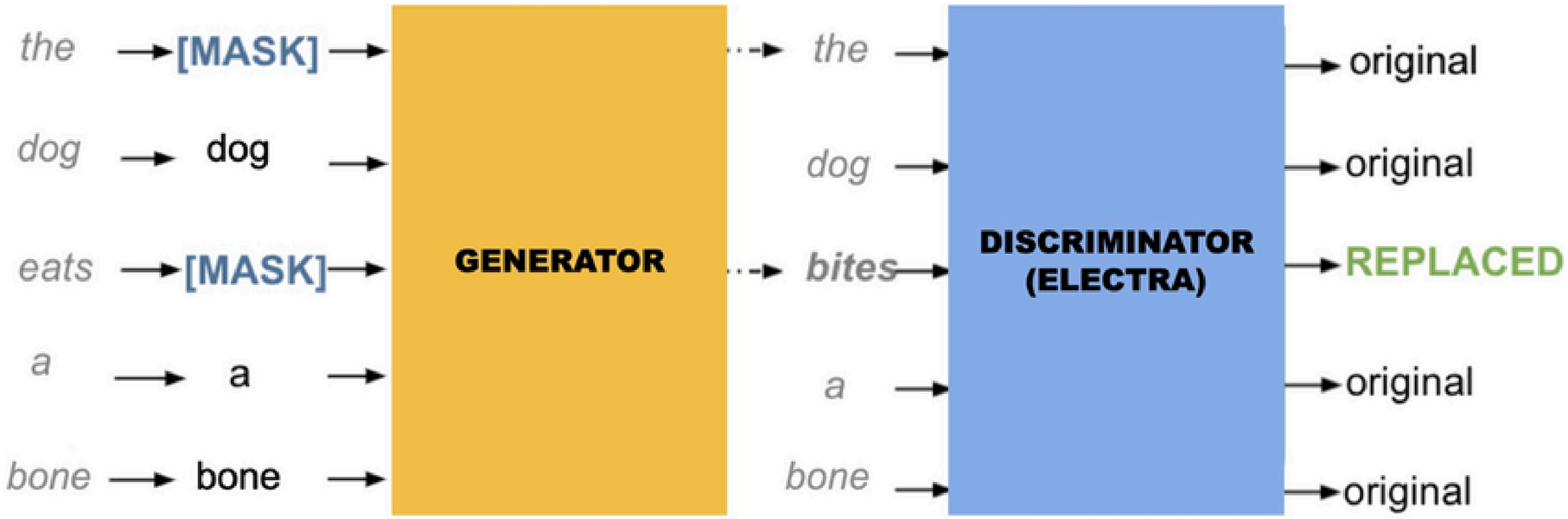


Cross-Layer Parameter Sharing

3. TinyBERT


Large-scale
Text Corpus
General
Distillation
General
TinyBERT
Task-specific
Distillation
Fine-tuned
TinyBERT
Task Dataset
Data Augmentation
Augmented
Task Dataset

4. ELECTRA


the
[MASK]
dog
dog
eats
[MASK]
a
a
bone
bone
GENERATOR
the
dog
bites
a
bone
DISCRIMINATOR (ELECTRA)
original
original
REPLACED
original
original

IV. Data Source

Analysis

- Basic email template: subject, content
- Various writing styles: formal, informal, malicious
- Various language patterns: sexually explicit, spam-like

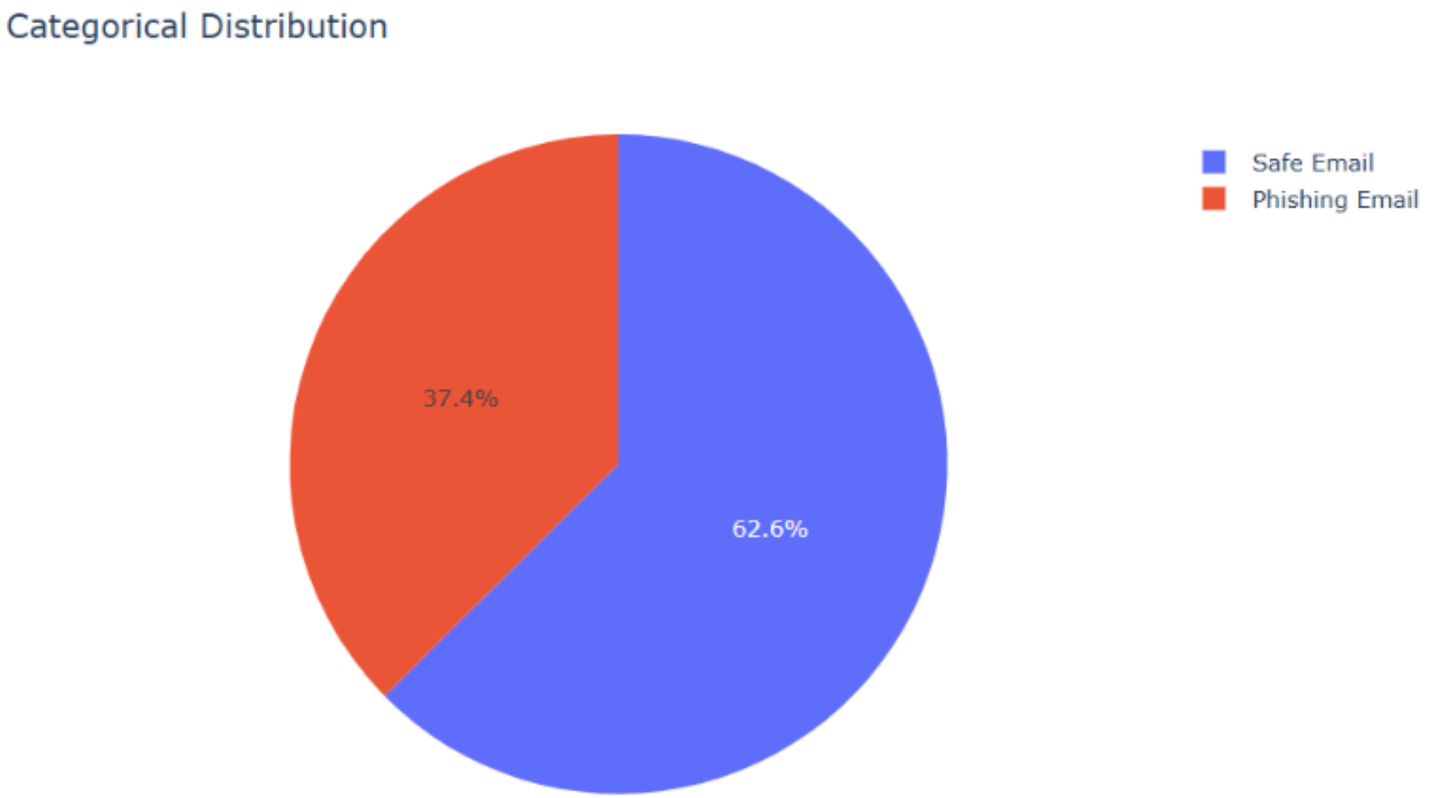
Analysis

```
22,"
Question?Do you want a different job?
Do you want to be your own boss?
Do you need extra income?
Do you need to start a new life?
Does your current job seem to go nowhere?If you answered yes to these questions, then here is your solution.We are a fortune 500 compa
looking
to a substantial income working from home.Thousands of individual are currently do this RIGHT NOW.
So if you are looking to be employed at home, with a career that will
provide you vast opportunities and a substantial income, please fill
out our online information request form here now:http://ter.netblah.com:27000To miss out on this opportunity, click herehttp://ter.net
",Phishing Email
```

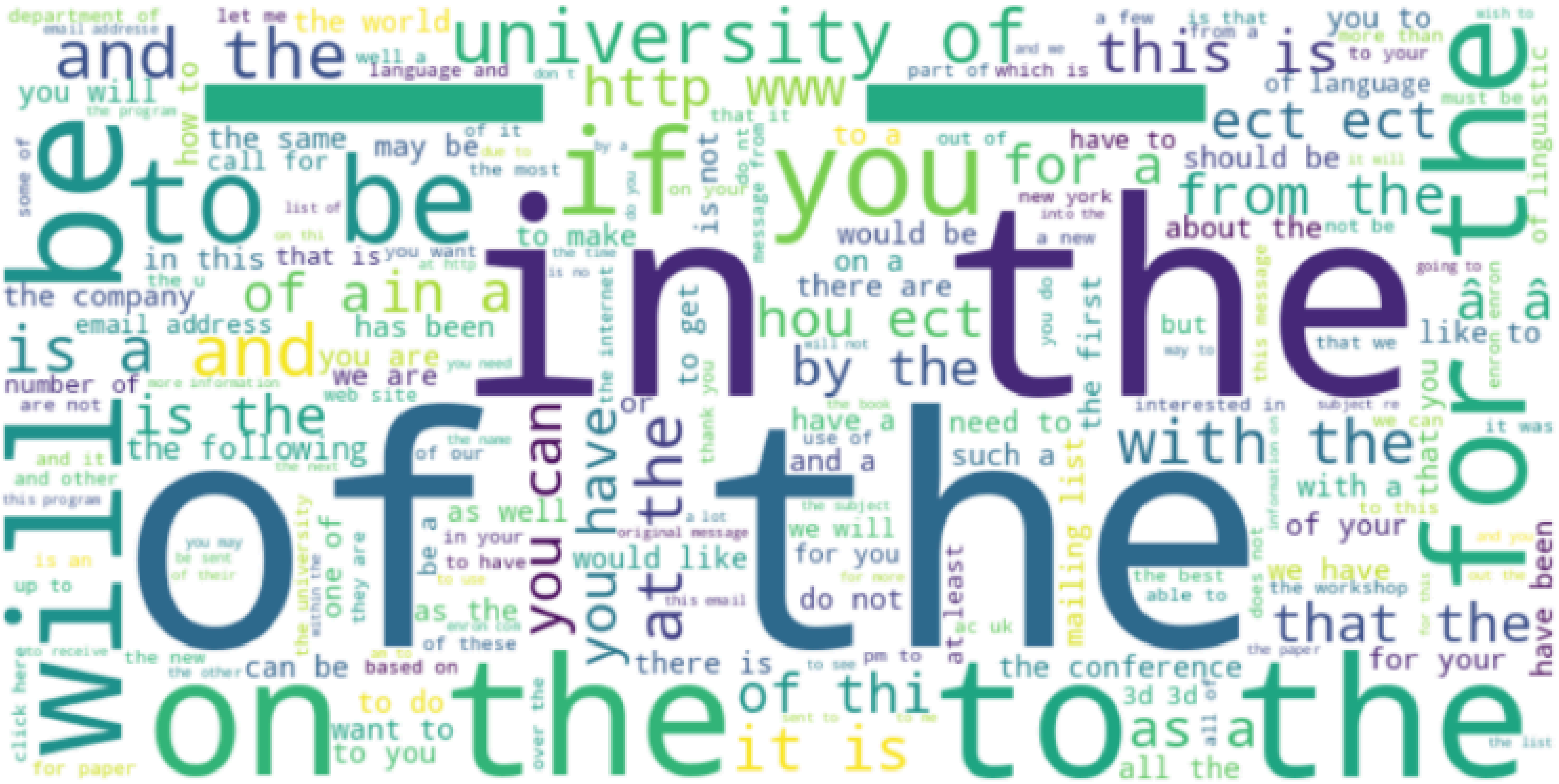
Analysis

```
6,"On Sun, Aug 11, 2002 at 11:17:47AM +0100, wintermute mentioned:
> > The impression I get from reading lkml the odd time is
> > that IDE has gone downhill since Andre Hedrick was
> > effectively removed as maintainer. Martin Dalecki seems
> > to have been unable to further development without
> > much breakage.
>
> Hmm... begs the question, why remove Handrick?
> If it ain't broke, don't fix it. See, the IDE subsystem is like the One Ring. It's kludginess, due to
having to support hundreds of dodgy chipsets & drives means that it is
inherintly evil. A few months of looking at the code can turn you sour.
Years of looking at it will turn you into an arsehole. They haven't found a hobbit that can code, so mortal humans have to
suffice. Kate
--
Irish Linux Users' Group: ilug@linux.ie
http://www.linux.ie/mailman/listinfo/ilug for (un)subscription information.
List maintainer: listmaster@linux.ie
",Safe Email
```

Analysis


Categorical Distribution
37.4%
62.6%
Safe Email
Phishing Email

Analysis


of the
in the
for the
will be
to be
if you
you
university of
and the
this is
http www
on the
to the
the
it is
ect ect
from the
with the
that the
by the
hou ect
and
you can
you have
at the
is a
is the
of a
in a
the following
the company
email address
has been
would be
there are
for a
should be
about the
the conference
can be
would like
we will
for you
do not
there is
need to
such a
have a
and a
of your
we have
the same
may be
to make
call for
in this
that is
you will
the world
let me
department of
of language
you to
number of
are not
as well
one of
to do
want to
to you
all the
as a
for your
have been
mailing list
the first
like to
with a
but
on a
of linguistic

Analysis


will
one
language
time
work
use
people
now
information
may
know
make
uneed
report
order
program
company
paper
new
way
ect ect
hou ect
e mail
s
linguistic
business
system
english
problem
see
thank
following
question
change
name
think
issue
said
number
word
many
want
first
day
available
call
message
book
free
used
send
product
well
case
mailing list
email address
example
university
provide
part
project
deal
lot
today
conference
reference
service
interest
include
interested
software
world
enron
check
received
help
much
made
data
year
area
sent
address
right
line
workshop
two
still
fact
give
result
contact
request
letter
internet
market
take
put
possible
application
email
position
discussion
week
click
found
based
note
come
using
given
point
development
cost
site
thing
account
good
file
abstract
mean
money
offer
read
seem
course
end
share
able
member
say
going
price
plan
held
receive
web site
find
follow
list
least
best
research
process
support
response
back
review
even
million
set
detail
look
term
place
english
might
department
3d 3d
formetc
new york
including

Preprocessing

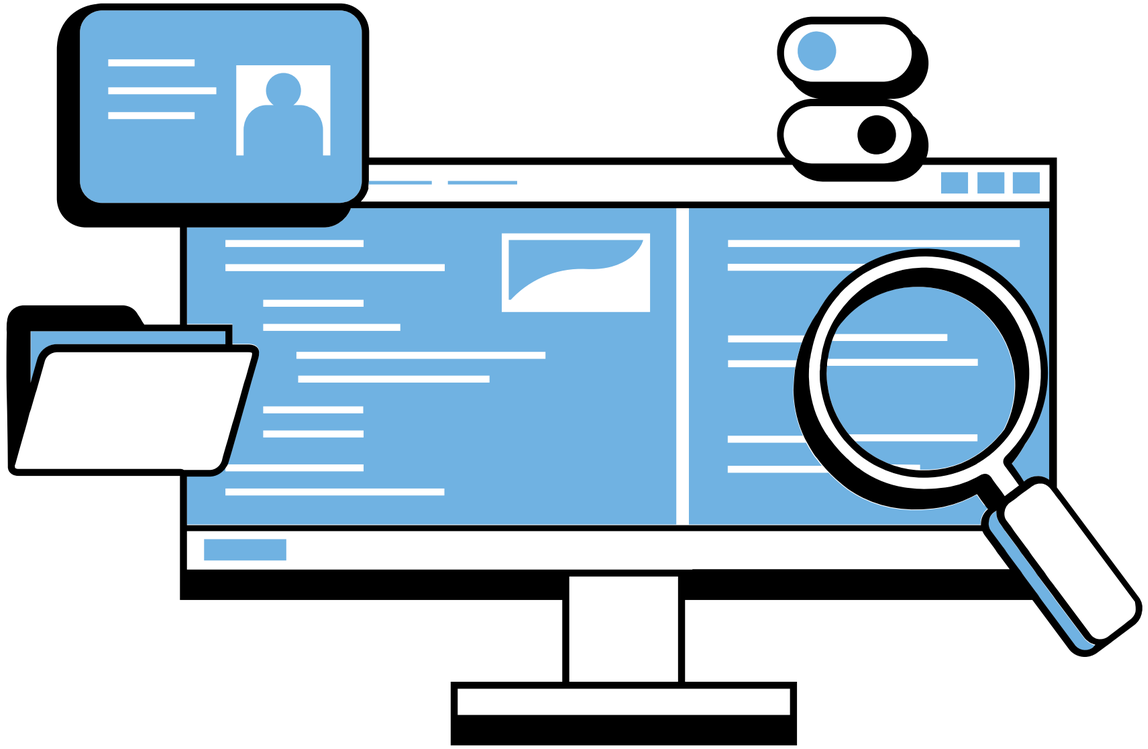
```python
def load_and_preprocess_kfold():
    df = pd.read_csv("Dataset/phishingEmail.csv")
    df.drop(["Unnamed: 0"], axis=1, inplace=True, errors="ignore")
    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)

    le = LabelEncoder()
    df["Email Type"] = le.fit_transform(df["Email Type"])

    def preprocess_text(text):
        text = re.sub(r'http\S+', '', text)  # Remove hyperlinks
        text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
        text = text.lower()  # Convert to lowercase
        text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
        return text

    df["processed_text"] = df["Email Text"].apply(preprocess_text)

    tf = TfidfVectorizer(stop_words="english", max_features=10000)
    X = tf.fit_transform(df["processed_text"])
    y = np.array(df['Email Type'])

    return df, X, y
```

Preprocessing

```python
def load_and_preprocess():
    df = pd.read_csv("Dataset/phishingEmail.csv")
    df.isnull().sum()
    df.drop(["Unnamed: 0"], axis=1, inplace=True, errors="ignore")
    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)

    def preprocess_text(text):
        text = re.sub(r'http\S+', '', text)  # Remove hyperlinks
        text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
        text = text.lower()  # Convert to lowercase
        text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
        return text

    df["Email Text"] = df["Email Text"].apply(preprocess_text)

    tf = TfidfVectorizer(stop_words="english", max_features=10000)
    feature_x = tf.fit_transform(df["Email Text"]).toarray()
    y_tf = np.array(df['Email Type'])


    x_train, x_test, y_train, y_test = train_test_split(feature_x, y_tf, train_size=0.8, random_state=0)
    return df, x_train, x_test, y_train, y_test
```

V. Implementation

1. Data Loading and Splitting
2. Evaluation Metrics and Visualization
3. Model Implementation



1. Data Loading and Splitting

```
df, X, y = load_and_preprocess_kfold()
```

```
df, x_train, x_test, y_train, y_test = load_and_preprocess()
```

```
for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):

    print(f"\nFold {fold + 1}/{k}")

    x_train, x_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]
```

2. Evaluation Metrics and Visualization

```
acc = accuracy_score(y_val, preds)
f1 = f1_score(y_val, preds)
recall = recall_score(y_val, preds)
precision = precision_score(y_val, preds)
loss = log_loss(y_val, probs)
error_rate = 1 - acc

true_probs = np.array([probs[i, label] for i, label in enumerate(y_val)])
rmse = np.sqrt(mean_squared_error(np.ones_like(true_probs), true_probs))

accs.append(acc)
f1s.append(f1)
recalls.append(recall)
precisions.append(precision)
losses.append(loss)
rmses.append(rmse)

print(f"Fold {fold}")
print(f"Accuracy      : {acc*100:.2f}%")
print(f"F1 Score      : {f1*100:.2f}%")
print(f"Recall        : {recall*100:.2f}%")
print(f"Precision     : {precision*100:.2f}%")
print(f"Log Loss      : {loss:.4f}")
print(f"Error Rate    : {error_rate*100:.2f}%")
print(f"RMSE          : {rmse:.4f}")
```

```
    cm = confusion_matrix(y_val, preds)
    ConfusionMatrixDisplay(cm, display_labels=["Phishing", "Safe"]).plot()
    plt.title(f"Fold {fold} Confusion Matrix")
    plt.show()
    print("-" * 50)

print("\nCross-Validation Summary:")
print(f"Average Accuracy   : {np.mean(accs)*100:.2f}%")
print(f"Average F1 Score   : {np.mean(f1s)*100:.2f}%")
print(f"Average Recall     : {np.mean(recalls)*100:.2f}%")
print(f"Average Precision  : {np.mean(precisions)*100:.2f}%")
print(f"Average RMSE       : {np.mean(rmses):.4f}")
print(f"Average Error Rate : {(1 - np.mean(accs))*100:.2f}%")
print(f"Average Log Loss : {(1 - np.mean(loss))*100:.2f}%")
```

2. Evaluation Metrics and Visualization

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Figure 25. Accuracy Formula

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

Figure 26. Precision Formula

2. Evaluation Metrics and Visualization

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

Figure 27. Recall Formula

Precision = Of all the things you caught, how many were actually fish?

Recall = Of all the fish in the sea, how many did you catch?

2. Evaluation Metrics and Visualization

$$\text{F1 score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Figure 28. F1 Score Formula

2. Evaluation Metrics and Visualization

$$LogLoss = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} x_{ij} * \ log(p_{ij})$$

Figure 29. Log Loss Formula

2. Evaluation Metrics and Visualization

Error Rate = 1 - Accuracy

Figure 30. Error Rate Formula

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

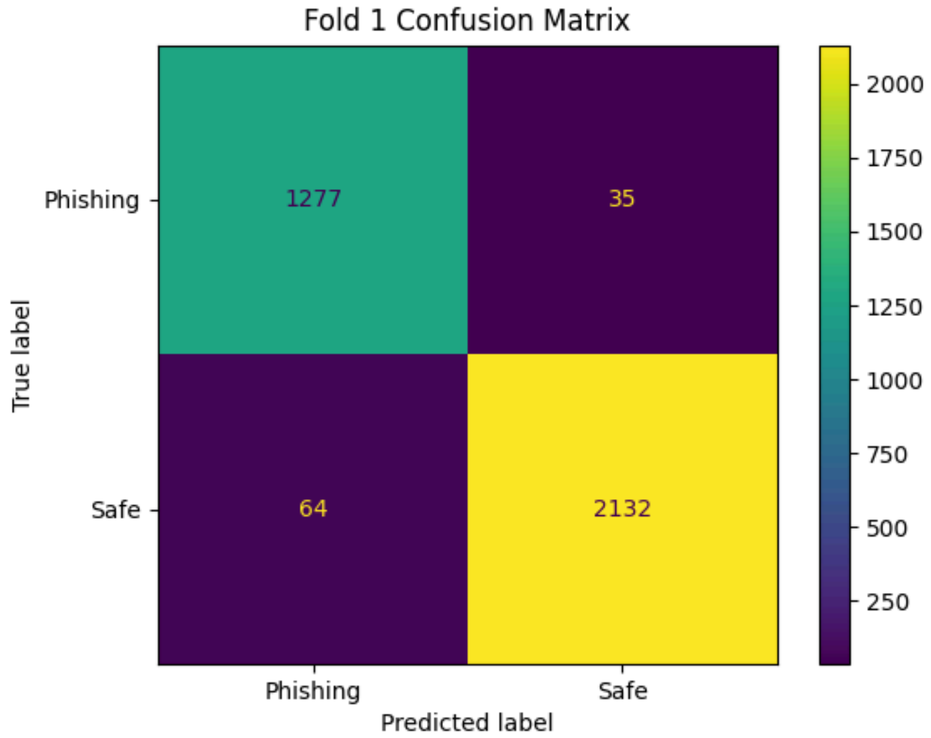
$\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ are predicted values

$y_1, y_2, \ldots, y_n$ are observed values

n is the number of observations

Figure 31. RMSE Formula

2. Evaluation Metrics and Visualization



2. Evaluation Metrics and Visualization

- Loss: Errors on the training set
- Accuracy: Correct predictions on the training set
- Validation Loss: Errors on the validation set
- Validation Accuracy: Correct predictions on the validation set

3. Implementation

I. CNN

```
model = Sequential([
    Dense(256, activation='relu', input_shape=(X.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=epochs,
    batch_size=batch_size,
    verbose=1
)

loss, acc = model.evaluate(x_val, y_val, verbose=0)
y_pred = model.predict(x_val).flatten()
y_pred_binary = (y_pred >= 0.5).astype(int)
```

II. ALBERT

```
model_name = "albert-base-v2"
tokenizer = AlbertTokenizer.from_pretrained(model_name)

train_texts = df["Email Text"].iloc[list(range(len(x_train)))].tolist()
test_texts = df["Email Text"].iloc[list(range(len(x_test)))].tolist()

train_encodings = tokenizer(train_texts, padding="max_length", truncation=True, max_length=64, return_tensors="tf")
test_encodings = tokenizer(test_texts, padding="max_length", truncation=True, max_length=64, return_tensors="tf")

train_dataset = tf.data.Dataset.from_tensor_slices((
    {"input_ids": train_encodings["input_ids"], "attention_mask": train_encodings["attention_mask"]},
    y_train
)).batch(16).prefetch(tf.data.AUTOTUNE)

test_dataset = tf.data.Dataset.from_tensor_slices((
    {"input_ids": test_encodings["input_ids"], "attention_mask": test_encodings["attention_mask"]},
    y_test
)).batch(16).prefetch(tf.data.AUTOTUNE)

model = TFAlbertForSequenceClassification.from_pretrained(model_name, num_labels=2)

for group in model.albert.encoder.albert_layer_groups[:2]:
    for layer in group.albert_layers:
        layer.trainable = False
```

II. ALBERT

```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"]
)

history = model.fit(
    train_dataset,
    validation_data=test_dataset,
    epochs=5
)

pred_logits = model.predict(test_dataset).logits
pred_probs = tf.nn.softmax(pred_logits, axis=1).numpy()
pred_labels = np.argmax(pred_probs, axis=1)
```

III. TinyBERT

```
model_name = "huawei-noah/TinyBERT_General_4L_312D"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

train_texts = df["Email Text"].iloc[list(range(len(x_train)))].tolist()
test_texts = df["Email Text"].iloc[list(range(len(x_test)))].tolist()

train_encodings = tokenizer(train_texts, padding=True, truncation=True, max_length=64, return_tensors="pt")
test_encodings = tokenizer(test_texts, padding=True, truncation=True, max_length=64, return_tensors="pt")

train_labels = torch.tensor(y_train)
test_labels = torch.tensor(y_test)

train_dataset = TensorDataset(train_encodings["input_ids"], train_encodings["attention_mask"], train_labels)
test_dataset = TensorDataset(test_encodings["input_ids"], test_encodings["attention_mask"], test_labels)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
optimizer = optim.AdamW(model.parameters(), lr=5e-5)
criterion = nn.CrossEntropyLoss()
```

IV. ELECTRA

```
model_name = "google/electra-small-discriminator"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

train_texts = df["Email Text"].iloc[list(range(len(x_train)))].tolist()
test_texts = df["Email Text"].iloc[list(range(len(x_test)))].tolist()

train_encodings = tokenizer(train_texts, padding=True, truncation=True, max_length=64, return_tensors="pt")
test_encodings = tokenizer(test_texts, padding=True, truncation=True, max_length=64, return_tensors="pt")

train_labels = torch.tensor(y_train.values)
test_labels = torch.tensor(y_test.values)

train_dataset = TensorDataset(train_encodings["input_ids"], train_encodings["attention_mask"], train_labels)
test_dataset = TensorDataset(test_encodings["input_ids"], test_encodings["attention_mask"], test_labels)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
optimizer = optim.AdamW(model.parameters(), lr=5e-5)
criterion = nn.CrossEntropyLoss()
```

VI. Experimental Result Analysis

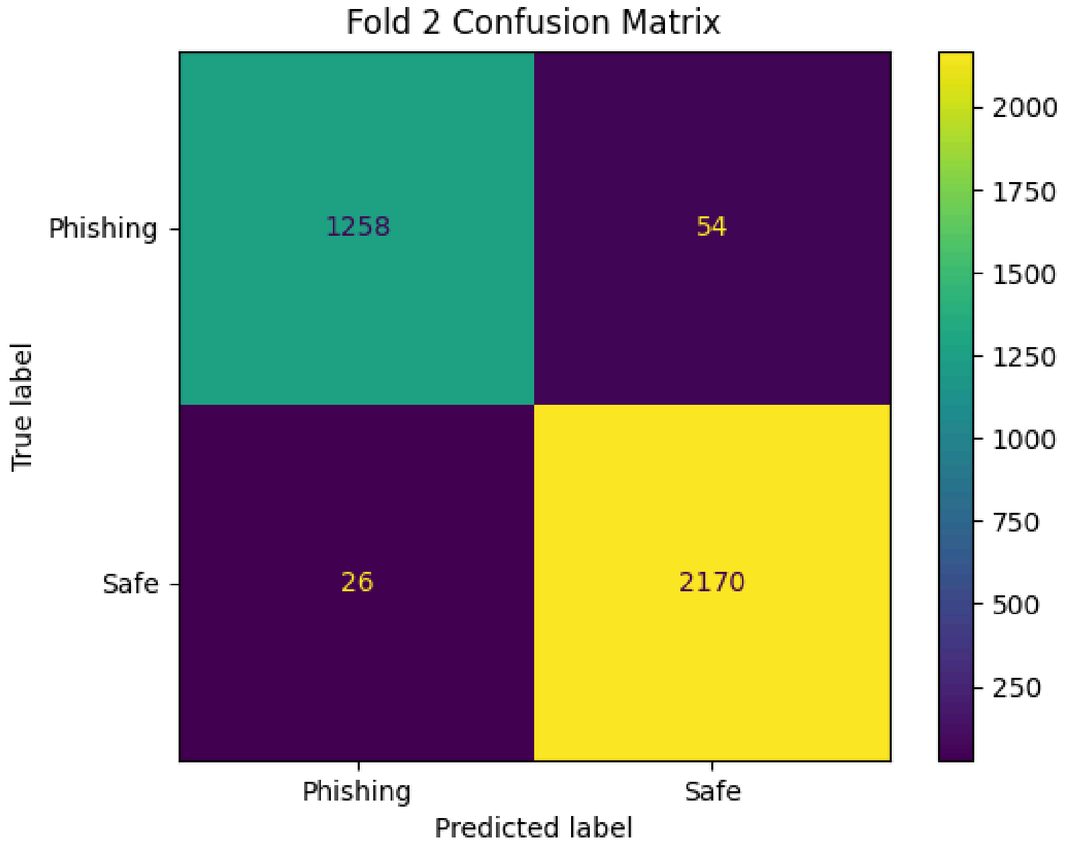
1. K-Fold Enhanced Models
2. Proposed Models

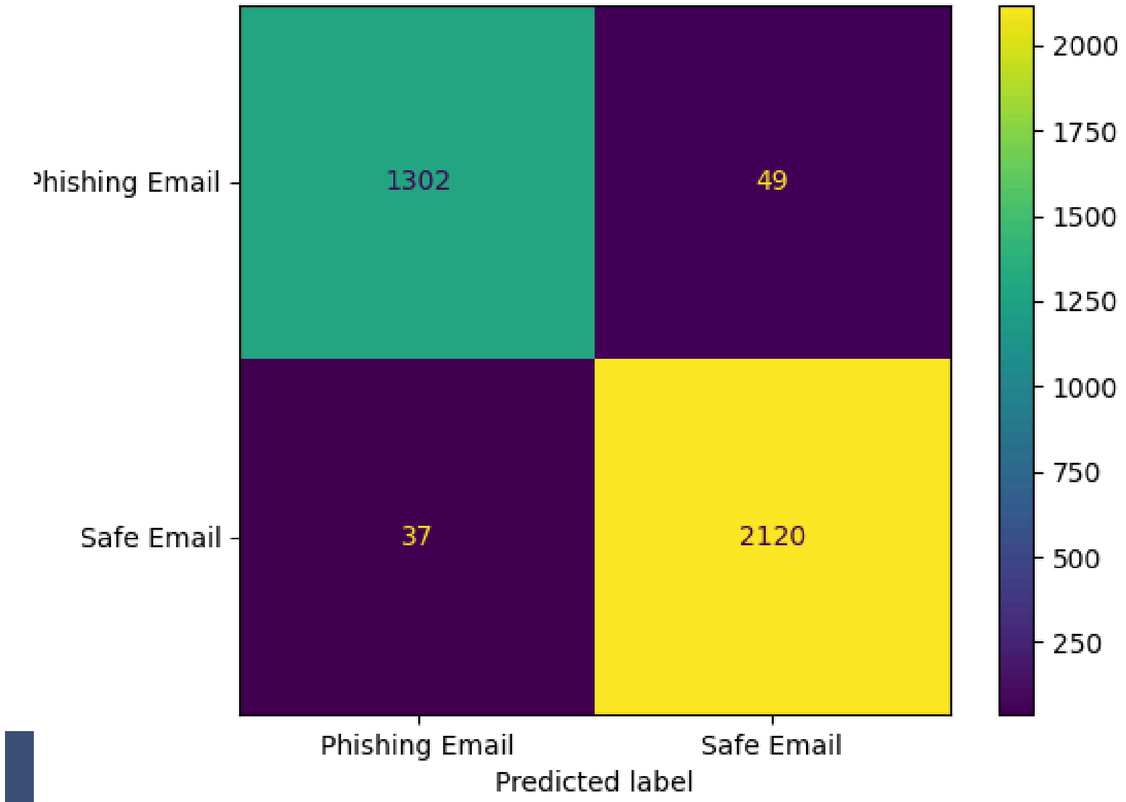
1. Naive Bayes

Model	Accuracy	Precision	Recall	F1 Score
K-Fold	0.9710	0.9721	0.9820	0.9770
Normal	0.9755	0.9724	0.9637	0.9680

Model	Log Loss	Error Rate	RMSE
K-Fold	0.0999	0.0290	0.1591
Normal	0.0930	0.0245	0.1516

1. Naive Bayes



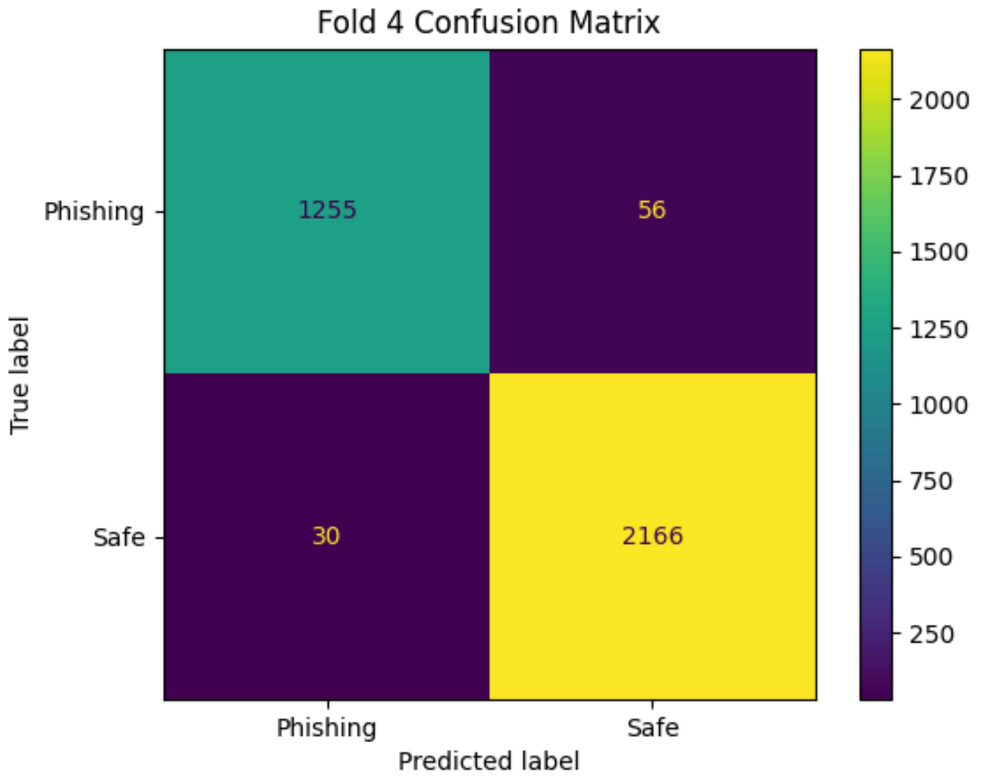


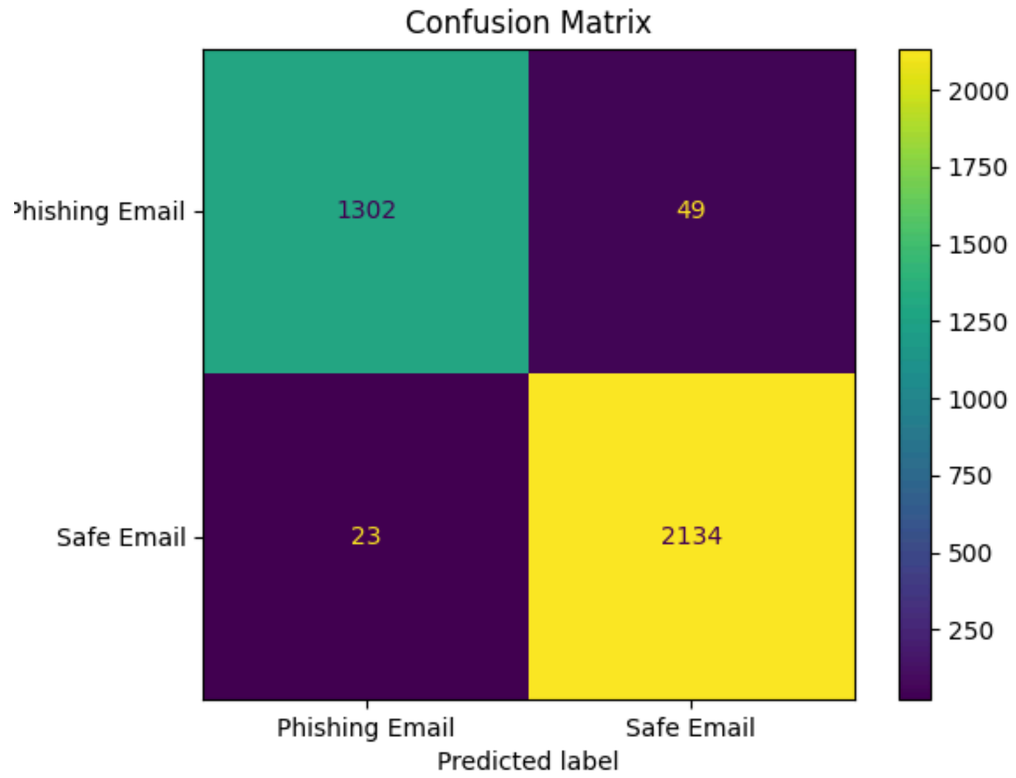
2. Logistic Regression

Model	Accuracy	Precision	Recall	F1 Score
K-Fold	0.9745	0.9735	0.9861	0.9797
Normal	0.9795	0.9826	0.9637	0.9731

Model	Log Loss	Error Rate	RMSE
K-Fold	0.1297	0.0255	0.1597
Normal	0.1228	0.0205	0.1623

2. Logistic Regression



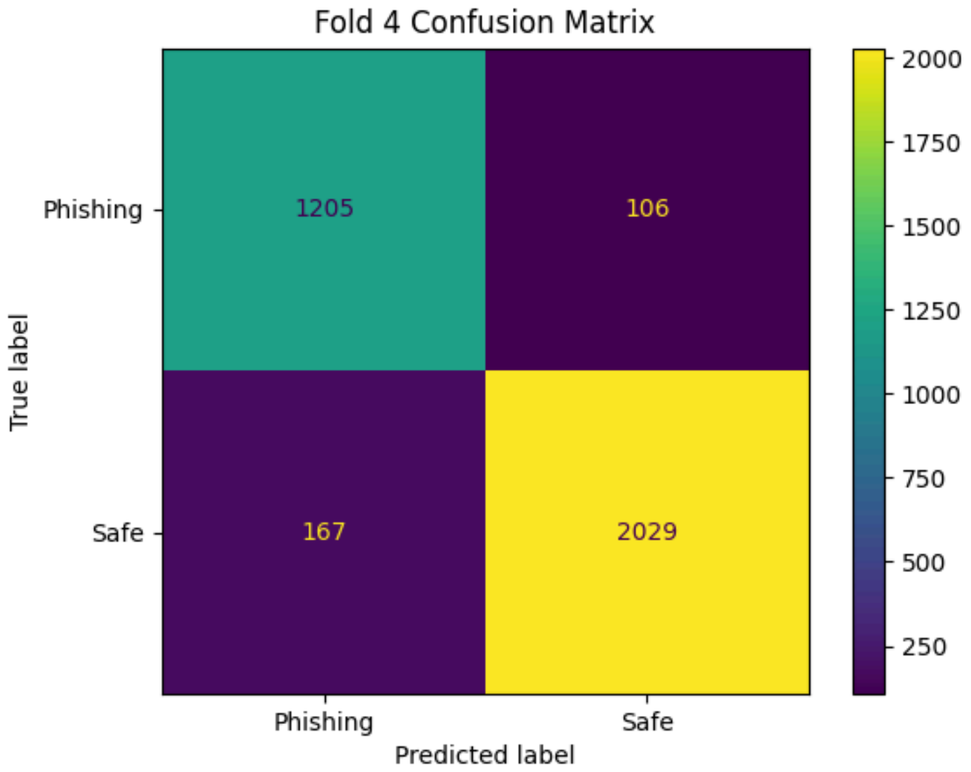


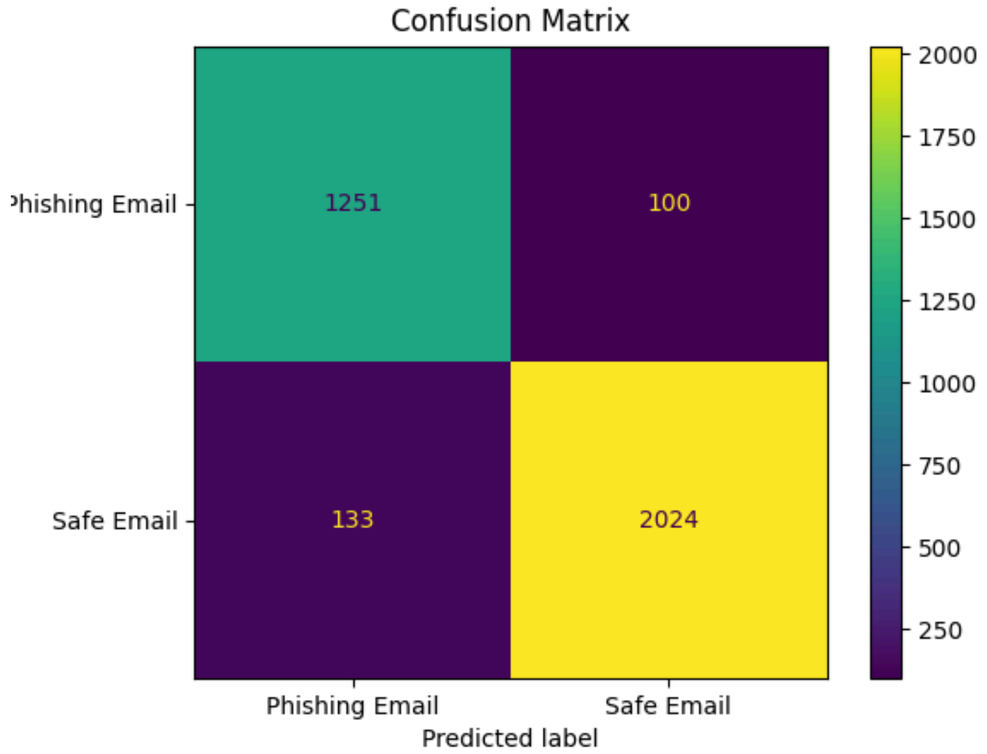
3. Decision Tree

Model	Accuracy	Precision	Recall	F1 Score
K-Fold	0.9229	0.9460	0.9300	0.9379
Normal	0.9336	0.9039	0.9260	0.9148

Model	Log Loss	Error Rate	RMSE
K-Fold	0.0258	0.0771	0.2776
Normal	0.0244	0.0664	0.2577

3. Decision Tree



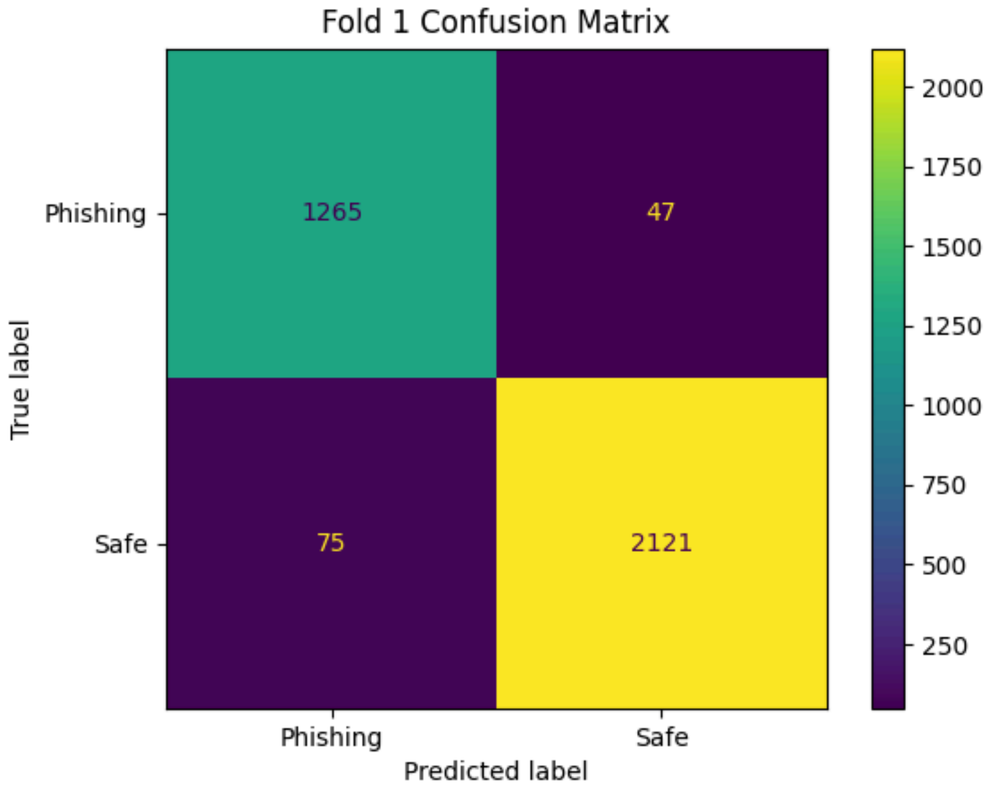


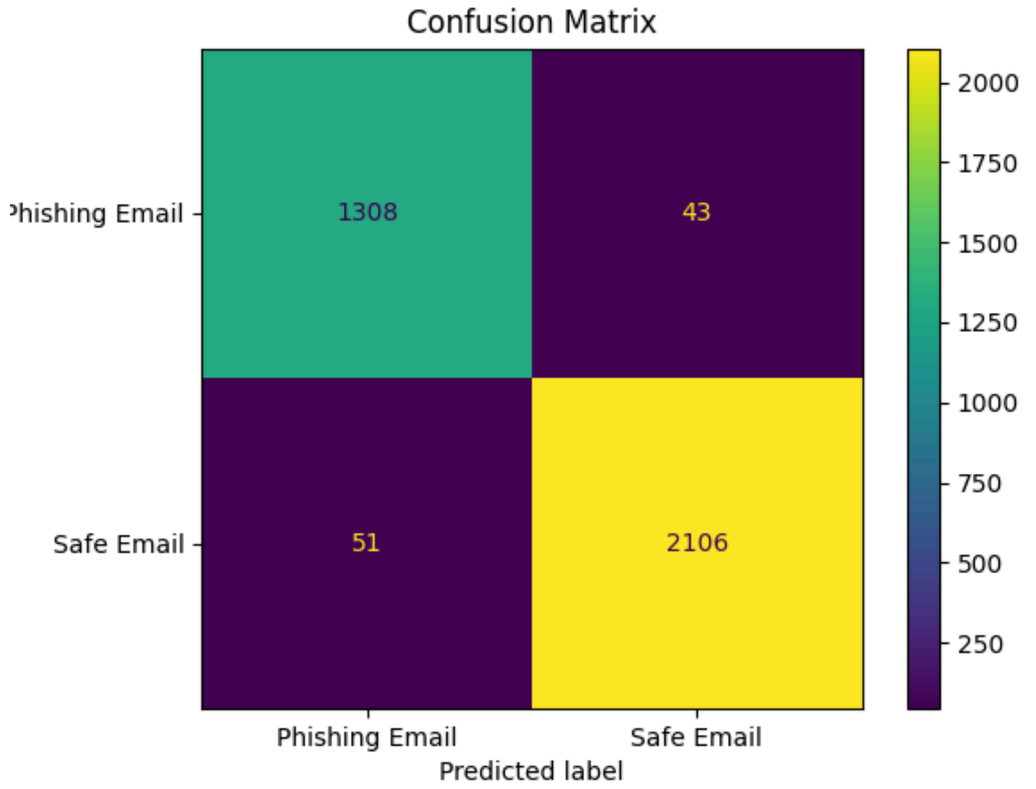
4. XGBoost

Model	Accuracy	Precision	Recall	F1 Score
K-Fold	0.9666	0.9777	0.9688	0.9732
Normal	0.9732	0.9800	0.9764	0.9782

Model	Log Loss	Error Rate	RMSE
K-Fold	0.0984	0.0334	0.1828
Normal	0.0910	0.0268	0.1637

4. XGBoost


Fold 1 Confusion Matrix
Phishing
Safe
True label
1265
47
75
2121
Phishing
Safe
Predicted label
2000
1750
1500
1250
1000
750
500
250

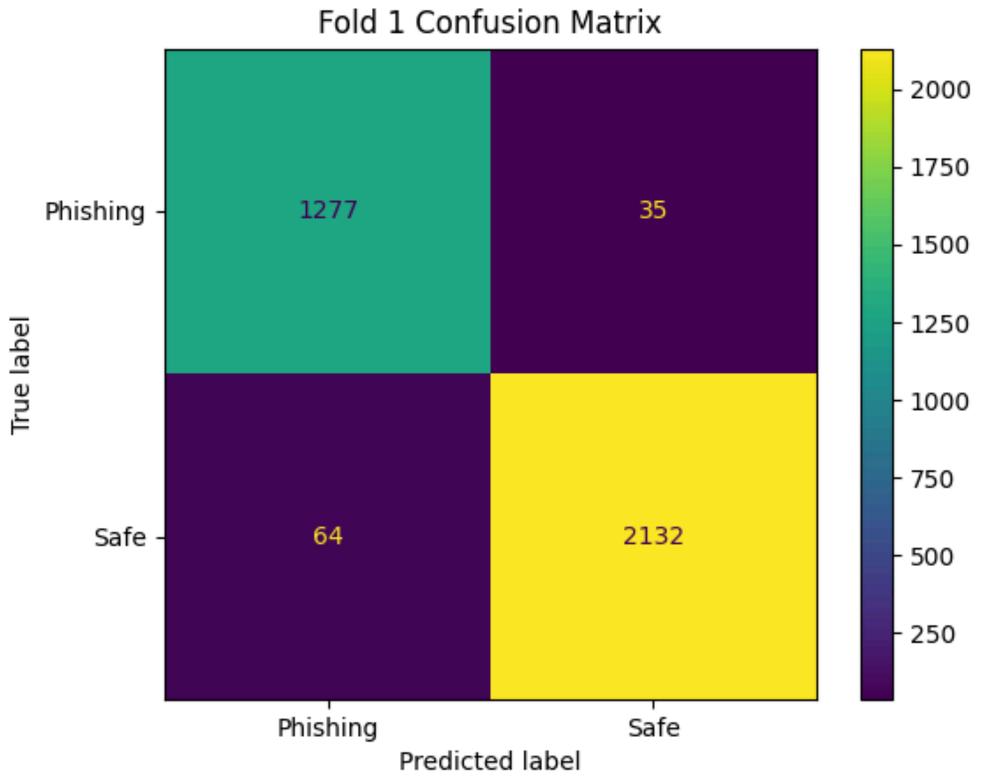

Confusion Matrix
Phishing Email
Safe Email
1308
43
51
2106
Phishing Email
Safe Email
Predicted label
2000
1750
1500
1250
1000
750
500
250

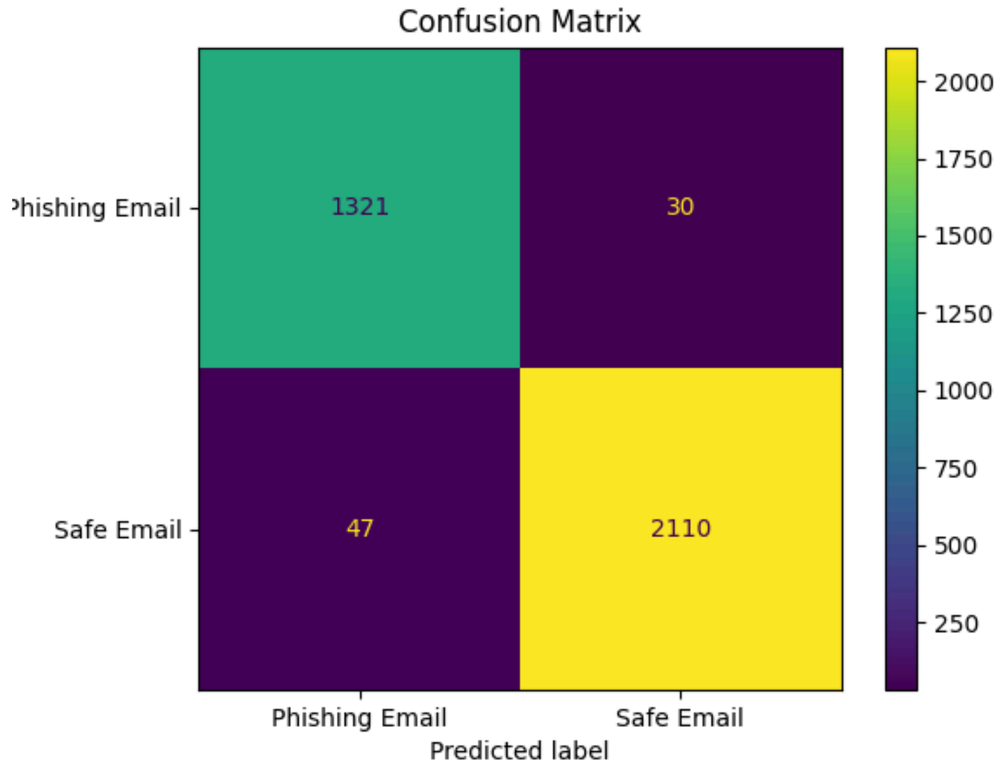
5. Random Forest

Model	Accuracy	Precision	Recall	F1 Score
K-Fold	0.9726	0.9844	0.9716	0.9780
Normal	0.9766	0.9846	0.9773	0.9809

Model	Log Loss	Error Rate	RMSE
K-Fold	0.1570	0.0274	0.1656
Normal	0.1466	0.0234	0.1529

5. Random Forest





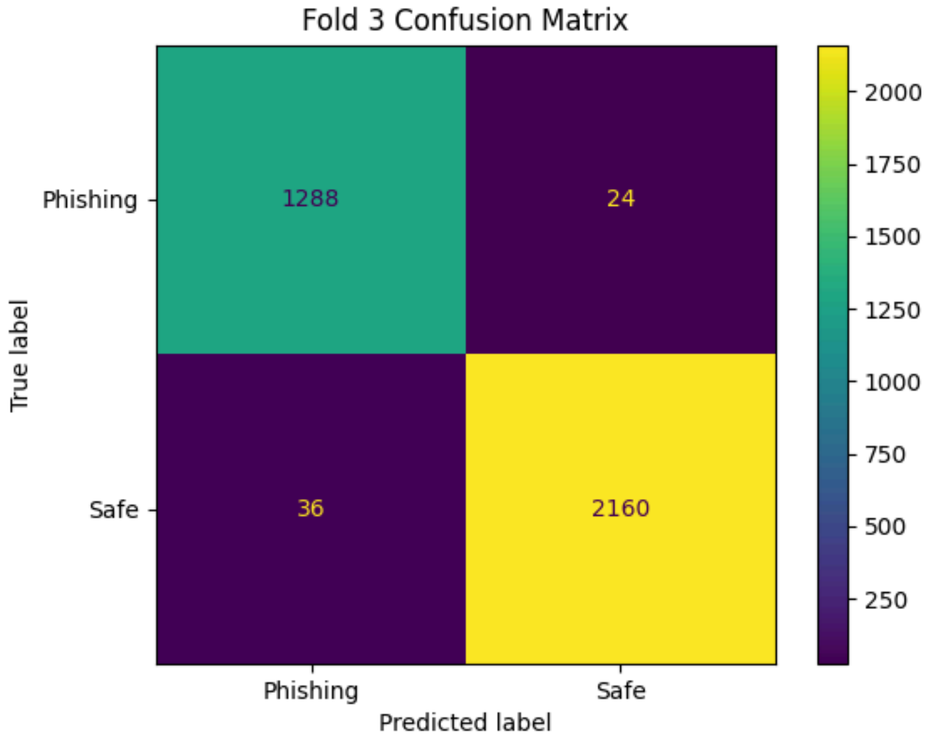
6. CNN

Fold	Accuracy	Precision	Recall	F1 Score
1	0.9812	0.9832	0.9868	0.9850
2	0.9846	0.9922	0.9832	0.9876
3	0.9829	0.9890	0.9836	0.9863
4	0.9798	0.9836	0.9841	0.9838
5	0.9769	0.9827	0.9804	0.9815
Average	0.9811	0.9861	0.9836	0.9849

6. CNN

Fold	Log Loss	Error Rate	RMSE
1	0.0783	0.0188	0.1249
2	0.0515	0.0154	0.1124
3	0.0581	0.0171	0.1156
4	0.0753	0.0202	0.1294
5	0.1038	0.0231	0.1418
Average	0.0734	0.0189	0.1248

6. CNN


Fold 3 Confusion Matrix
Phishing
Safe
True label
1288
24
36
2160
Phishing
Safe
Predicted label
2000
1750
1500
1250
1000
750
500
250

7. ALBERT

Epoch	Loss	Validation Loss	Accuracy	Validation Accuracy
1	0.6719	0.6665	0.6177	0.6149
2	0.6678	0.6778	0.6225	0.6149
3	0.6666	0.6672	0.6212	0.6149
4	0.6675	0.6719	0.6254	0.6149
5	0.6688	0.6679	0.6213	0.6149
Final	0.6679		0.6149	

8. TinyBERT

Epoch	Loss	Validation Loss	Accuracy	Validation Accuracy
1	0.0414	0.0418	0.6286	0.6149
2	0.0413	0.0419	0.6289	0.6149
3	0.0413	0.0418	0.6289	0.6149
4	0.0412	0.0419	0.6289	0.6149
5	0.0408	0.0434	0.6324	0.5841
Final	0.0434		0.5841	

9. ELECTRA

Epoch	Loss	Validation Loss	Accuracy	Validation Accuracy
1	0.0414	0.6272	0.0418	0.6149
2	0.0413	0.6289	0.0418	0.6149
3	0.0413	0.6289	0.0419	0.6149
4	0.0413	0.6289	0.0418	0.6149
5	0.0413	0.6289	0.0420	0.6149
Final	0.6289		0.6149	

Conclusion

Summary

- K-fold Recommend: Naive Bayes, Logistic Regression, CNN
- K-fold Not required: Decision Tree, XGBoost, Random Forest
- Unsuitable model: ALBERT, TinyBERT, ELECTRA

Improvement

- More balanced dataset
- Improve preprocessing function
- Apply hybrid model

THANK YOU

For your attention