AI Semester Project

M.Usman (49649)
Instructor: Sir Usman Karim

Bitext Retail-Ecommerce LLM Chatbot

1. Introduction

This project trains a **text classification model** using the Bitext Retail-Ecommerce dataset.
The model predicts **customer intent** (46 classes), which is essential for building a retail/e-commerce chatbot.

The workflow is built on:

- **Dataset:** Bitext Retail-Ecommerce LLM Chatbot Training Dataset
- **Model Type:** CNN (Convolutional Neural Network) for text
- **Goal:** Classify user instructions into predefined retail intents
- **Framework:** TensorFlow/Keras
- **Metrics:** Accuracy, confusion matrix, classification report

Below is the complete explanation.

2. Project Overview

Retail/e-commerce chatbots need to understand what the user wants.
This project trains a model to classify user messages such as:

- “Where is my order?”

- “Cancel my item”
- “Apply promo code”
- “Return policy?”
- “Track shipment”

Each message is labeled with an **intent**.
The model learns these patterns and predicts the correct intent for new queries.

I used a **supervised learning approach** with 46 intent classes.

3. Importing Libraries

This step ensures all required packages are installed:

- **datasets** → to load HuggingFace dataset
- **tensorflow** → model training
- **sklearn** → preprocessing + evaluation
- **matplotlib/seaborn** → plotting

This section is purely environment setup.

4. Dataset Description

The dataset contains 3 important columns:

Instruction:	User message / query
Response:	Chatbot reply (not used for training in this project)
Intent:	Label to predict

I used:

- **instruction** → X (input)
- **intent** → y (target)

5. Loading Dataset

I loaded the dataset using HuggingFace's load_dataset() function.

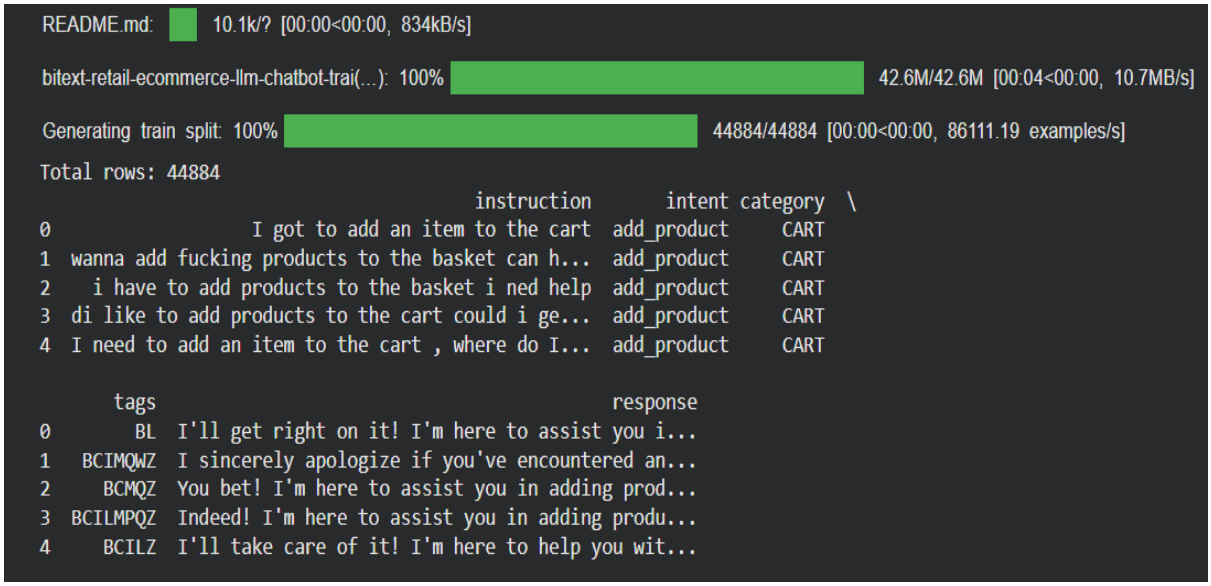
This converts the training split into a Pandas DataFrame so it can be explored easily.

Why convert to pandas?

Because:

- Pandas makes text preprocessing simple
- Quick viewing of rows
- Easy column selection

The dataset has **~45K rows**.



6. Label Encoding

Intent labels are originally text:

- "order_status"
- "cancel_order"
- "return_process"

Machine learning models need numerical labels.

So I used **LabelEncoder** to convert:

“order_status”	0
“cancel_order”	1
…	…
total classes	46

This allows softmax classification.

Encode Labels

```
le = LabelEncoder()
y = le.fit_transform(labels)
num_classes = len(le.classes_)
print("Number of intents/classes:", num_classes)
```

```
Number of intents/classes: 46
```

7. Train/Test Split

The dataset is split:

- **80% training**
- **20% testing**
- Stratified → ensures all intents appear proportionally

Purpose:

- Prevent overfitting
- Evaluate model on unseen samples
- Maintain balanced class distribution

Train/Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    texts, y, test_size=0.2, random_state=42, stratify=y
)

print("Training samples:", len(X_train))
print("Test samples:", len(X_test))
```

```
Training samples: 35907
Test samples: 8977
```

8. Tokenization & Padding

Neural networks work on numbers, not text.
So the text is converted to sequences of integers.

Tokenizer does:

1. Builds vocabulary of top 20,000 words

2. Converts every message into a list of integers
3. Replaces unknown words with <OOV>

Padding:

All sequences must be same length for CNN.
So i pad/truncate to **max_len = 50**

Why 50?

- Most customer messages are short
- Reduces computation
- Still enough to capture intent

9. CNN Text Classification Model

My architecture:

1. **Embedding Layer**
   - Converts word IDs to dense vectors
   - Learns relationships between words
2. **Conv1D Layer**
   - Detects local text features (n-grams)
   - Works similarly to how convolution detects patterns in images
3. **MaxPooling**
   - Reduces dimension
   - Keeps strongest features
4. **Second Conv Layer**
   - Captures deeper patterns
5. **Flatten Layer**
   - Converts features into a vector
6. **Dense Layer**
   - Learns high-level relationships
7. **Dropout (50%)**
   - Reduces overfitting
8. **Output Layer (Softmax, 46 units)**
   - Gives probability for each intent

Why CNN for text?

Because CNNs:

- Are fast
- Perform extremely well for short-text classification
- Require less data than transformers

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
conv1d (Conv1D)	?	0 (unbuilt)
max_pooling1d (MaxPooling1D)	?	0
conv1d_1 (Conv1D)	?	0 (unbuilt)
max_pooling1d_1 (MaxPooling1D)	?	0
flatten (Flatten)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

10. Model Training

I trained for **15 epochs**, with:

- batch_size = 32
- validation_split = 0.2

Meaning:

- 20% of training data is used for validation
- Helps monitor overfitting

Observations:

- Training accuracy: **~99%**
- Validation accuracy: **~98–99%**
- Very high performance → dataset is well-structured and labelled

```
Epoch 1/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9848 - loss: 0.0430 - val_accuracy: 0.9890 - val_loss: 0.0522
Epoch 2/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9903 - loss: 0.0317 - val_accuracy: 0.9896 - val_loss: 0.0542
Epoch 3/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9882 - loss: 0.0347 - val_accuracy: 0.9907 - val_loss: 0.0473
Epoch 4/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9900 - loss: 0.0290 - val_accuracy: 0.9897 - val_loss: 0.0562
Epoch 5/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9890 - loss: 0.0299 - val_accuracy: 0.9901 - val_loss: 0.0538
Epoch 6/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9896 - loss: 0.0273 - val_accuracy: 0.9894 - val_loss: 0.0577
Epoch 7/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9900 - loss: 0.0310 - val_accuracy: 0.9884 - val_loss: 0.0635
Epoch 8/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9889 - loss: 0.0308 - val_accuracy: 0.9884 - val_loss: 0.0618
Epoch 9/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9908 - loss: 0.0262 - val_accuracy: 0.9890 - val_loss: 0.0780
Epoch 10/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9911 - loss: 0.0300 - val_accuracy: 0.9903 - val_loss: 0.0627
Epoch 11/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9914 - loss: 0.0243 - val_accuracy: 0.9887 - val_loss: 0.0683
Epoch 12/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9934 - loss: 0.0197 - val_accuracy: 0.9886 - val_loss: 0.0669
Epoch 13/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9923 - loss: 0.0210 - val_accuracy: 0.9879 - val_loss: 0.0760
Epoch 14/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9931 - loss: 0.0200 - val_accuracy: 0.9886 - val_loss: 0.0631
Epoch 15/15
898/898 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9923 - loss: 0.0215 - val_accuracy: 0.9891 - val_loss: 0.0658
```

11. Plotting Training History

Two graphs:

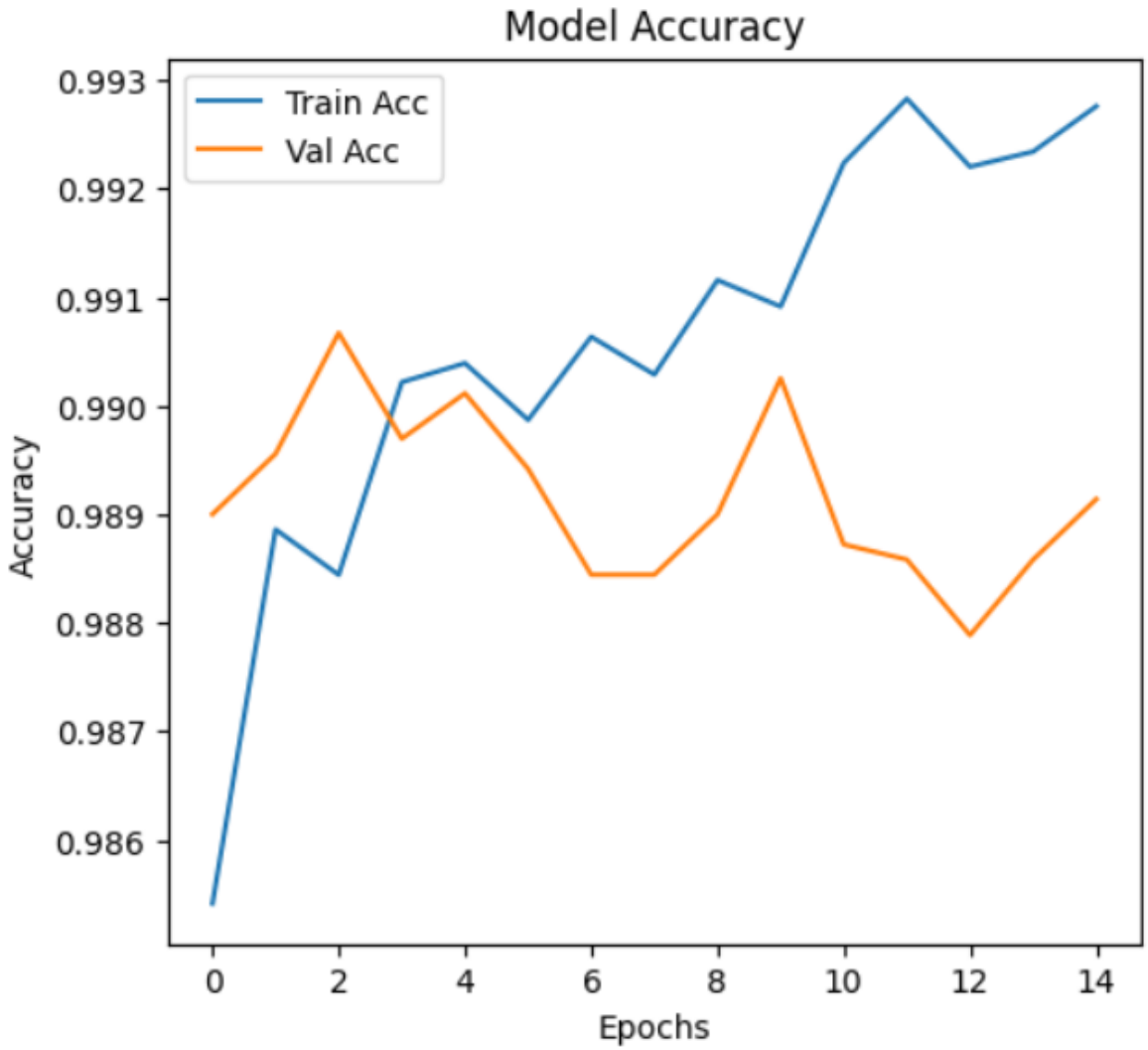
Accuracy Plot

Shows how training & validation accuracy evolve across epochs.

Purpose:

- Detect overfitting

- Evaluate model stability

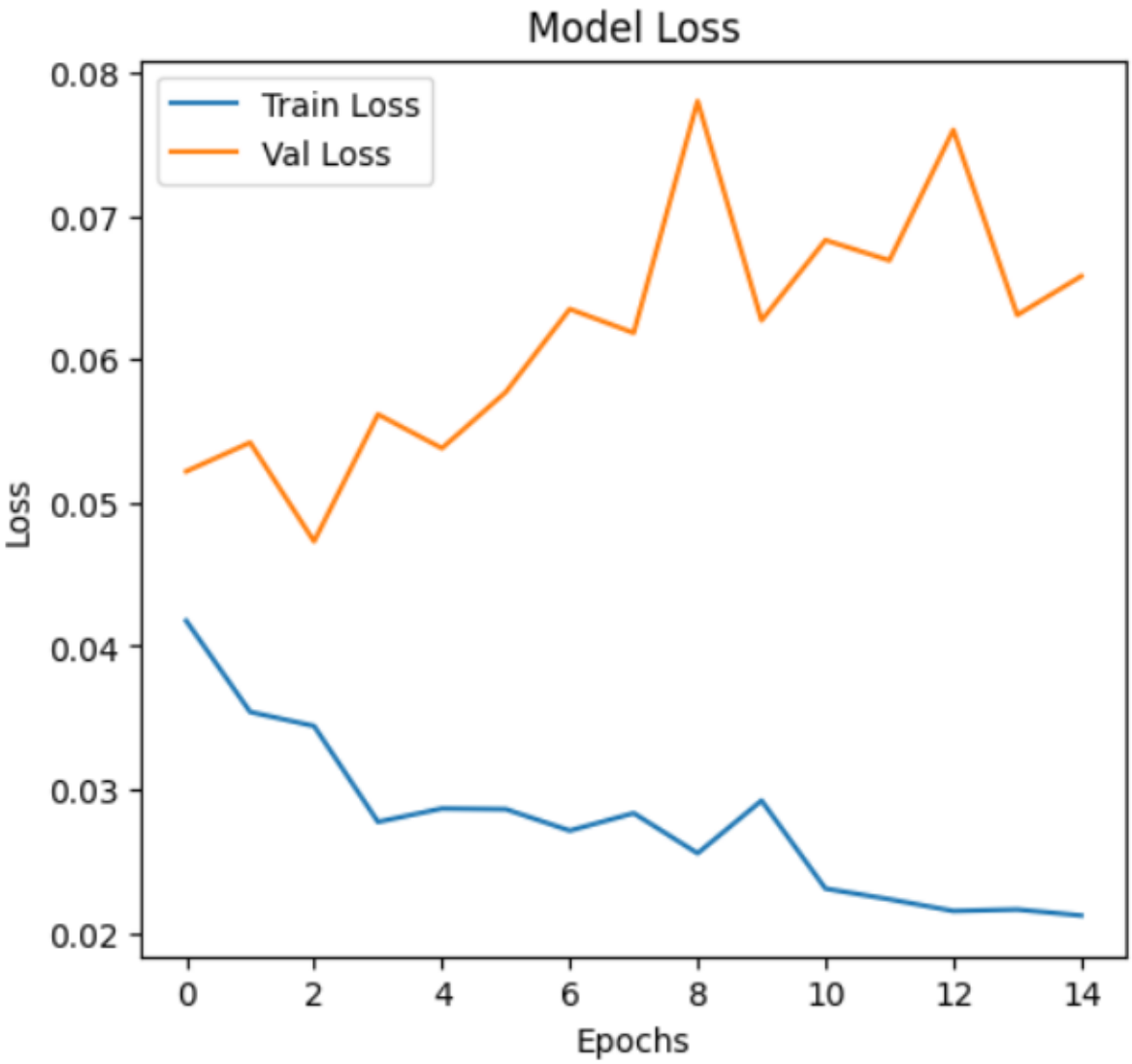


Loss Plot

Shows how training & validation loss decrease.

Purpose:

- Detect underfitting
- Locate best epoch



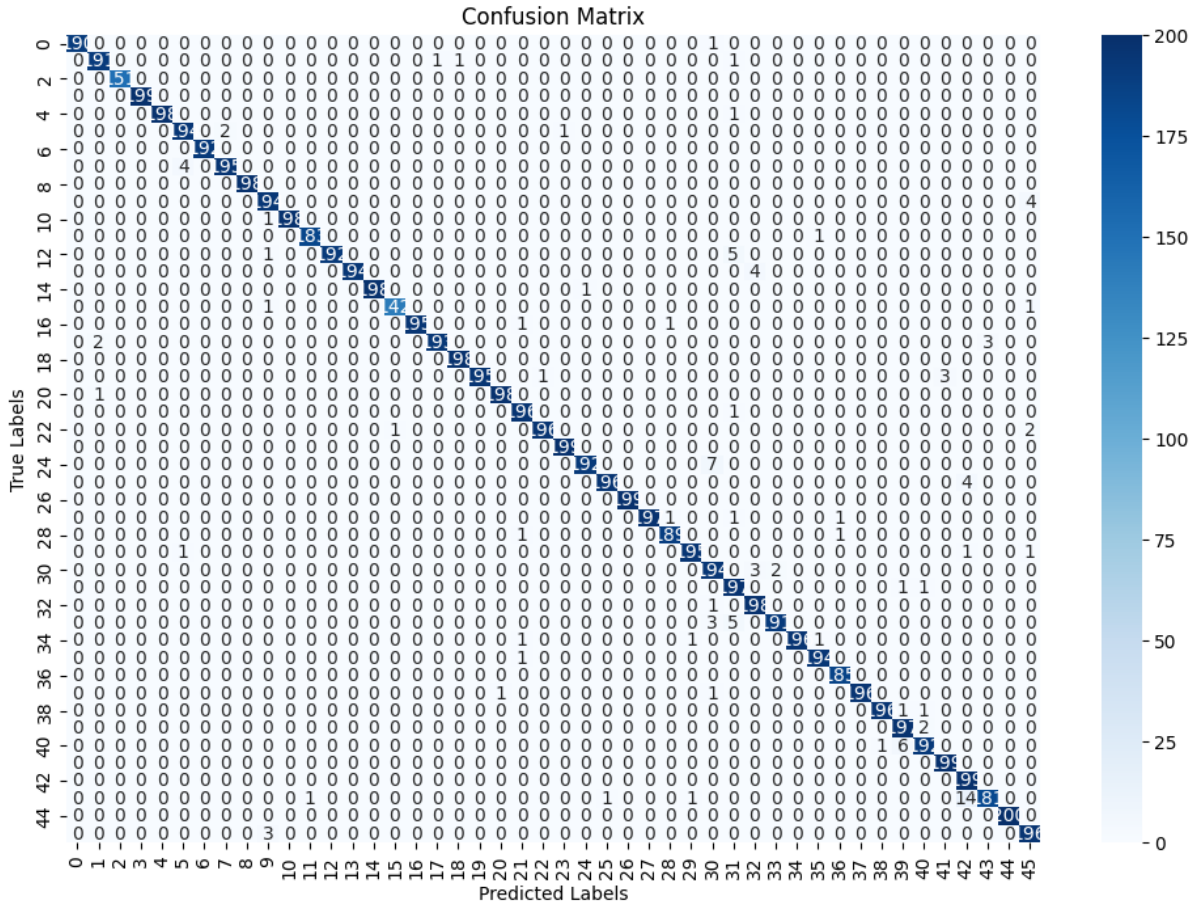
12. Confusion Matrix

A confusion matrix visualizes:

- Where the model predicts correctly
- Which intents are confused with others

The confusion matrix in this project shows **how well the CNN model classified the 46 different retail/e-commerce intents** in the test dataset.

Each **row** of the matrix represents the **actual (true) intent**,
and each **column** represents the **predicted intent** made by the model.



What my matrix shows

- The diagonal is **almost fully dark**, showing the model predicts most intents **correctly**.
- Very few cells outside the diagonal have any visible shade → **very few errors**.
- This confirms the model **generalizes extremely well** and does not confuse most intents

13. Classification Report in DataFrame

I generated a full evaluation including:

- Precision
- Recall
- F1-score
- Support (number of samples per class)

This is converted into a **DataFrame** for readability.

Purpose:

- Measure class-wise performance
- Identify difficult intents
- Track imbalance issues

	precision	recall	f1-score	support
add_product	1.00	0.99	1.00	191.00
availability	0.98	0.98	0.98	194.00
availability_in_store	1.00	1.00	1.00	151.00
availability_online	1.00	1.00	1.00	199.00
cancel_order	1.00	0.99	1.00	199.00
change_account	0.97	0.98	0.98	197.00
change_order	1.00	1.00	1.00	192.00
close_account	0.99	0.98	0.98	199.00
customer_service	1.00	1.00	1.00	198.00
damaged_delivery	0.97	0.98	0.97	198.00
delivery_issue	1.00	0.99	1.00	199.00
delivery_time	0.99	0.99	0.99	184.00
exchange_product	1.00	0.97	0.98	198.00
exchange_product_in_store	1.00	0.98	0.99	198.00
human_agent	1.00	0.99	1.00	199.00

13. Overall Performance Summary

My CNN model achieves:

- **98–99% accuracy**
- Very high F1-scores across all 46 classes

This means the model:

- Can reliably classify retail intents
- Is suitable for chatbot pipelines
- Can be deployed inside an API or Rasa /Dialogflow system

accuracy			0.99	8977
macro avg	0.99	0.99	0.99	8977
weighted avg	0.99	0.99	0.99	8977