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.

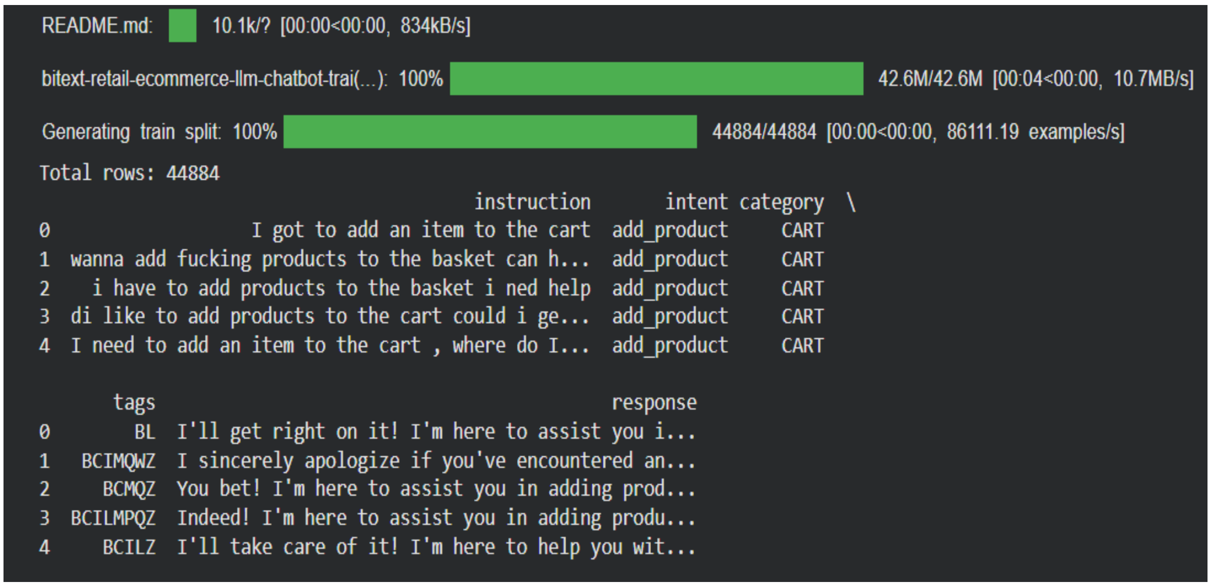
This converts the training split into a Pandas DataFrame so it can be explored easily.

Why convert to pandas?

Because:

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

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



```
README.md:  10.1k/? [00:00<00:00, 834kB/s]
bitext-retail-ecommerce-llm-chatbot-trai(…): 100%  42.6M/42.6M [00:04<00:00, 10.7MB/s]
Generating train split: 100%  44884/44884 [00:00<00:00, 86111.19 examples/s]
Total rows: 44884
                                         instruction       intent category  \
0                 I got to add an item to the cart  add_product     CART
1  wanna add fucking products to the basket can h...  add_product     CART
2    i have to add products to the basket i ned help  add_product     CART
3  di like to add products to the cart could i ge...  add_product     CART
4  I need to add an item to the cart , where do I...  add_product     CART

      tags                                         response
0       BL  I'll get right on it! I'm here to assist you i...
1  BCIMQWZ  I sincerely apologize if you've encountered an...
2    BCMQZ  You bet! I'm here to assist you in adding prod...
3 BCILMPQZ  Indeed! I'm here to assist you in adding produ...
4    BCILZ  I'll take care of it! I'm here to help you wit...
```

6. Label Encoding

Intent labels are originally text:

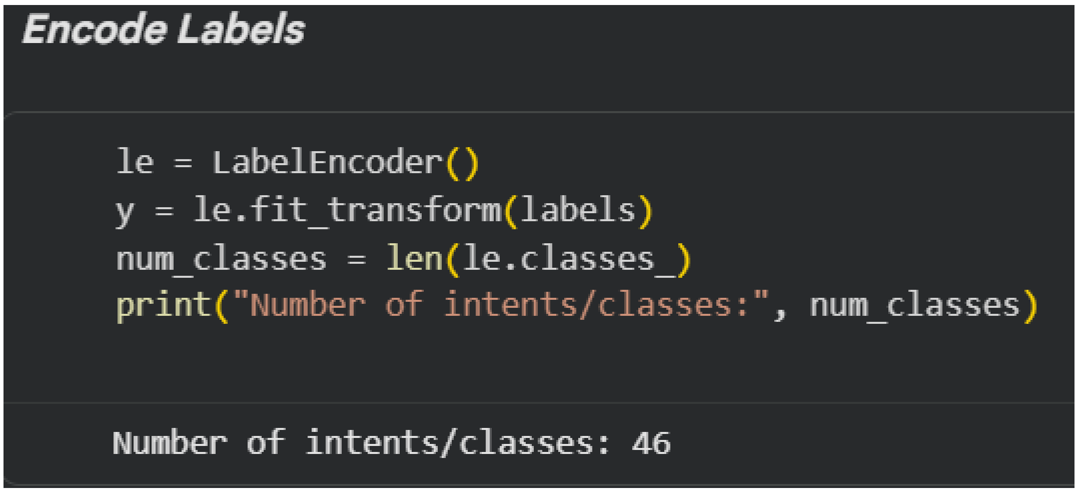
- "order_status"
- "cancel_order"
- "return_process"

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
```

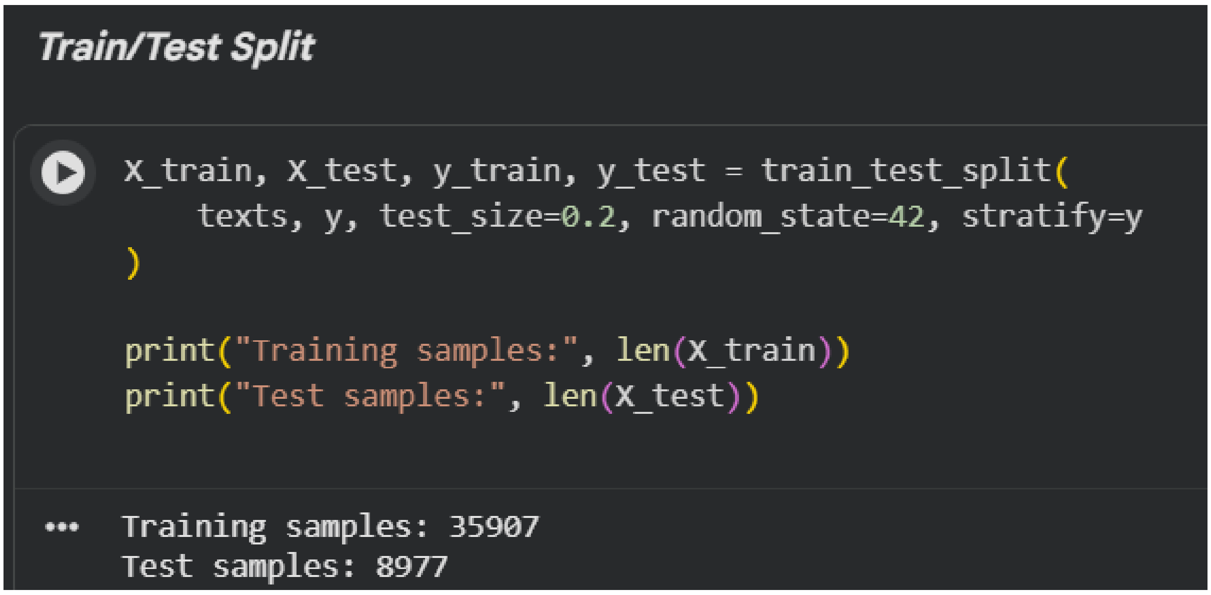
7. Train/Test Split

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

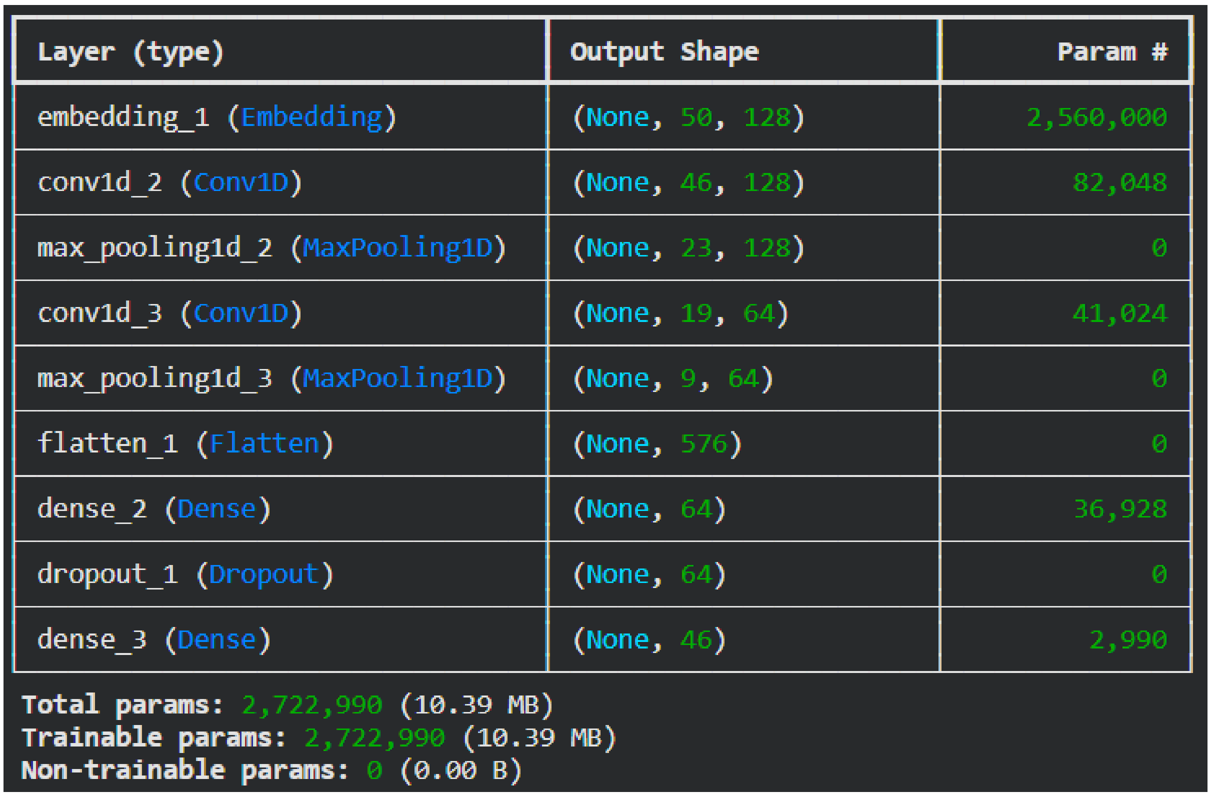
My architecture:

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_1 (Embedding)          │ (None, 50, 128)       │      2,560,000 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ conv1d_2 (Conv1D)                │ (None, 46, 128)       │         82,048 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ max_pooling1d_2 (MaxPooling1D)   │ (None, 23, 128)       │              0 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ conv1d_3 (Conv1D)                │ (None, 19, 64)        │         41,024 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ max_pooling1d_3 (MaxPooling1D)   │ (None, 9, 64)         │              0 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ flatten_1 (Flatten)              │ (None, 576)           │              0 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ dense_2 (Dense)                  │ (None, 64)            │         36,928 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ dropout_1 (Dropout)              │ (None, 64)            │              0 │
├──────────────────────────────────┼───────────────────────┼────────────────┤
│ dense_3 (Dense)                  │ (None, 46)            │          2,990 │
└──────────────────────────────────┴───────────────────────┴────────────────┘
 Total params: 2,722,990 (10.39 MB)
 Trainable params: 2,722,990 (10.39 MB)
 Non-trainable params: 0 (0.00 B)
```

10. Model Training

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

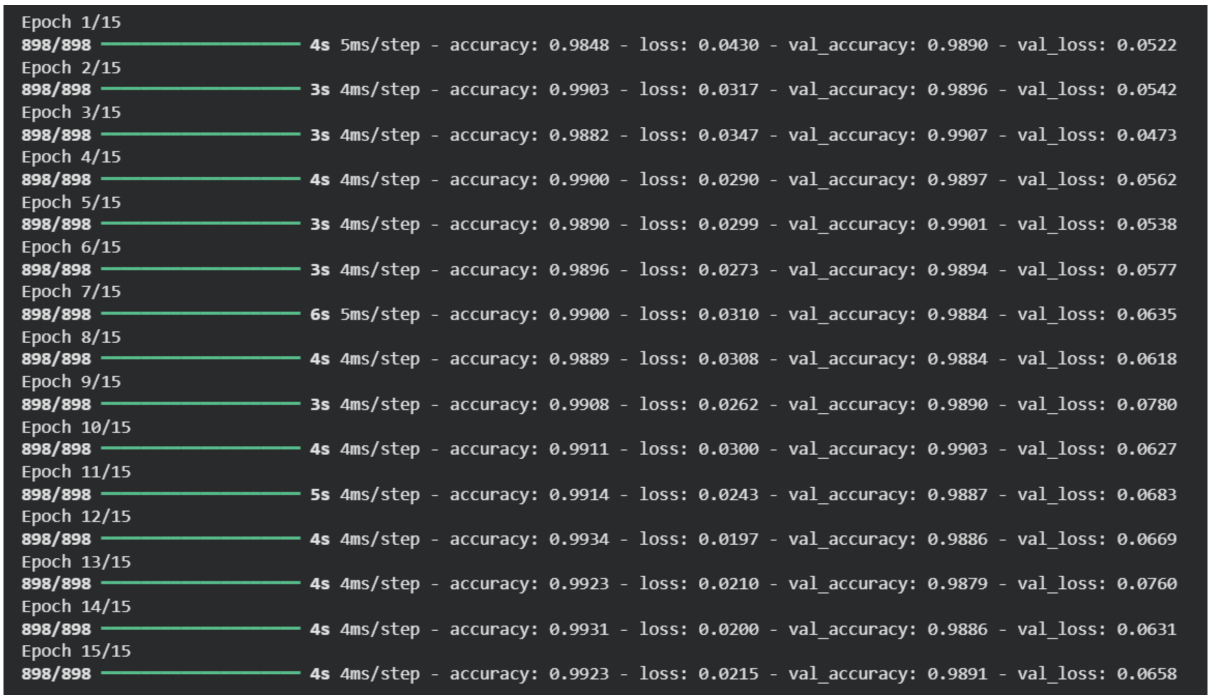
- batch_size = 32
- validation_split = 0.2

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:

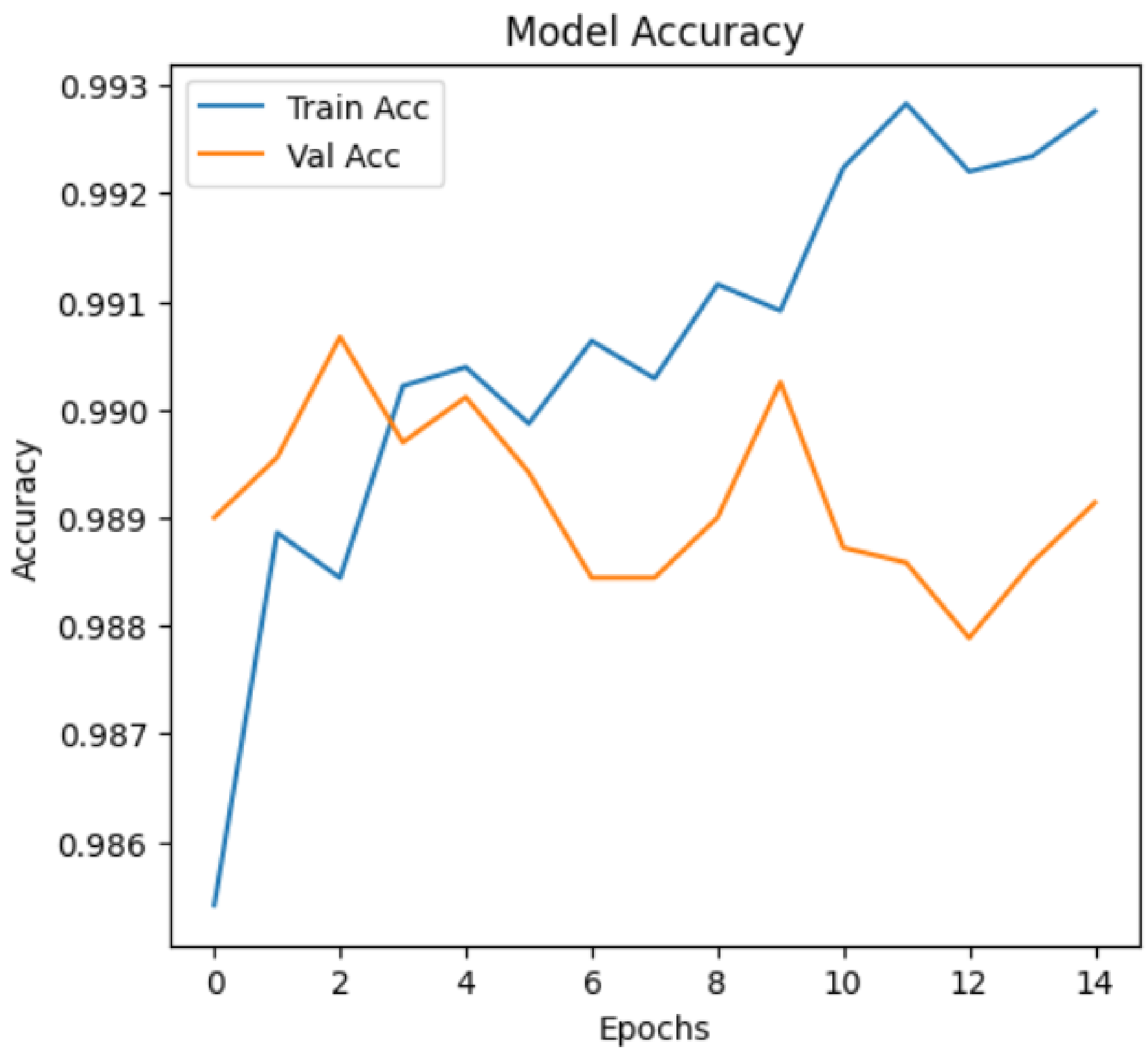
Accuracy Plot

Shows how training & validation accuracy evolve across epochs.

Purpose:

- Detect overfitting

- Evaluate model stability

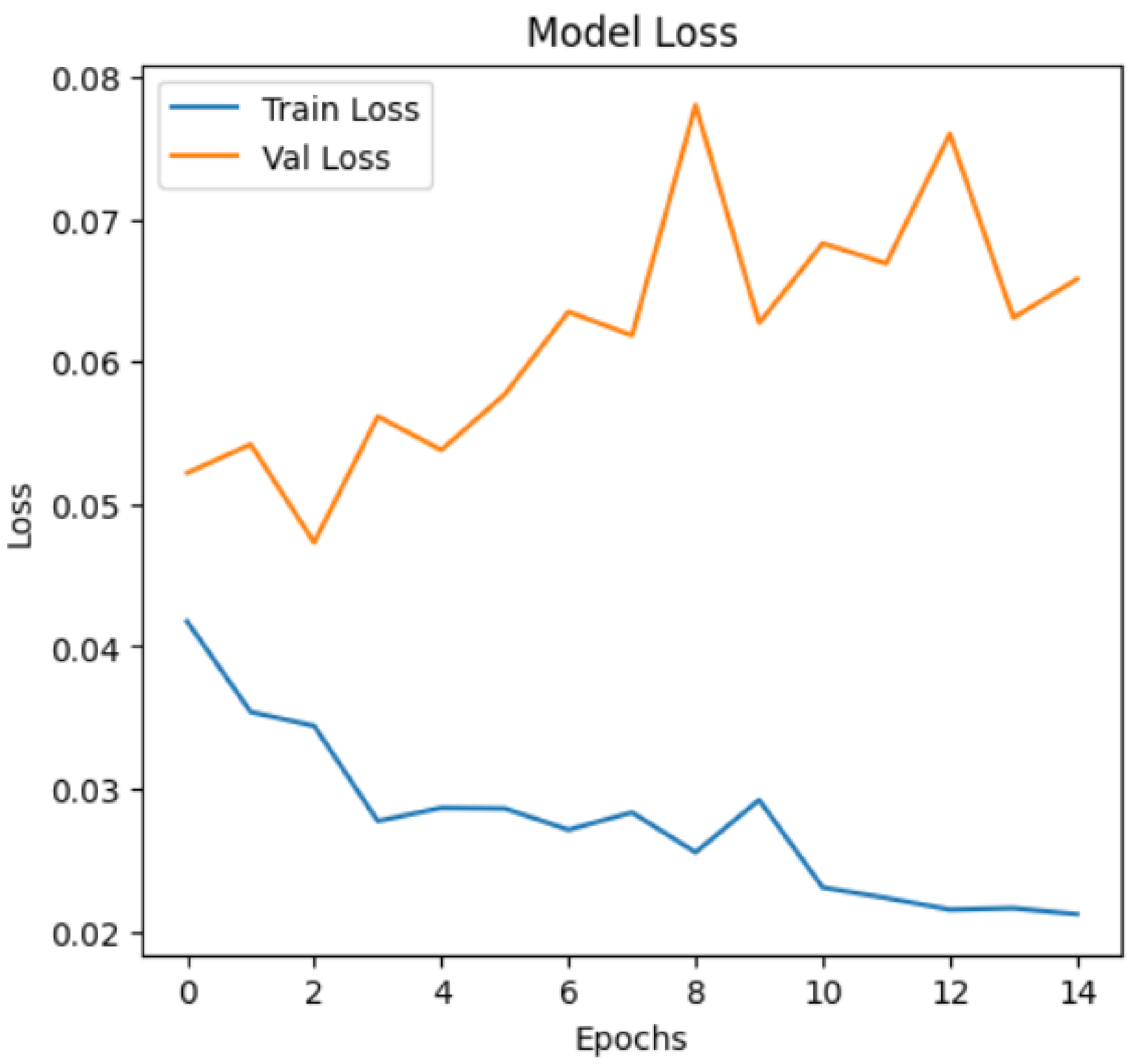


Loss Plot

Shows how training & validation loss decrease.

Purpose:

- Detect underfitting
- Locate best epoch



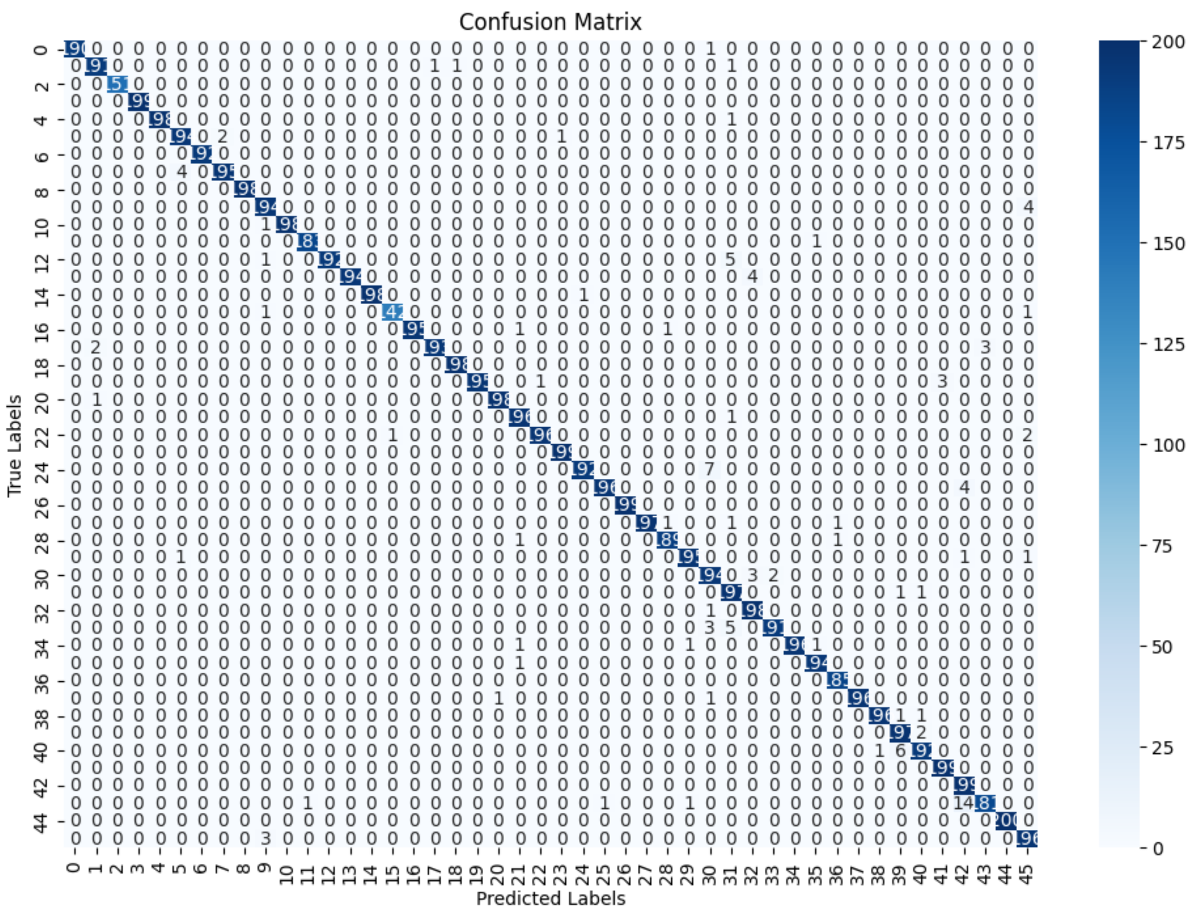
12. Confusion Matrix

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

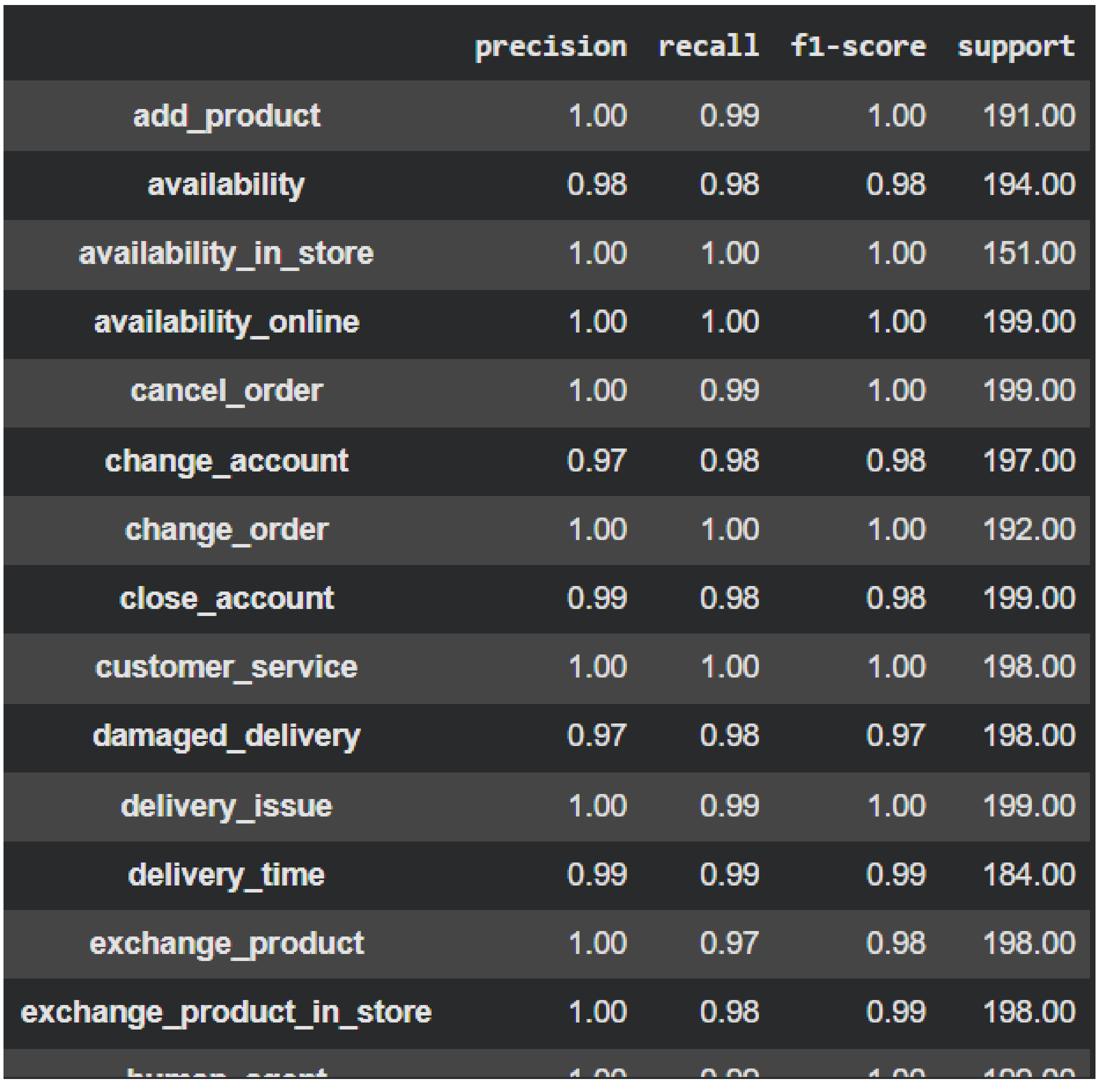
I generated a full evaluation including:

- 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



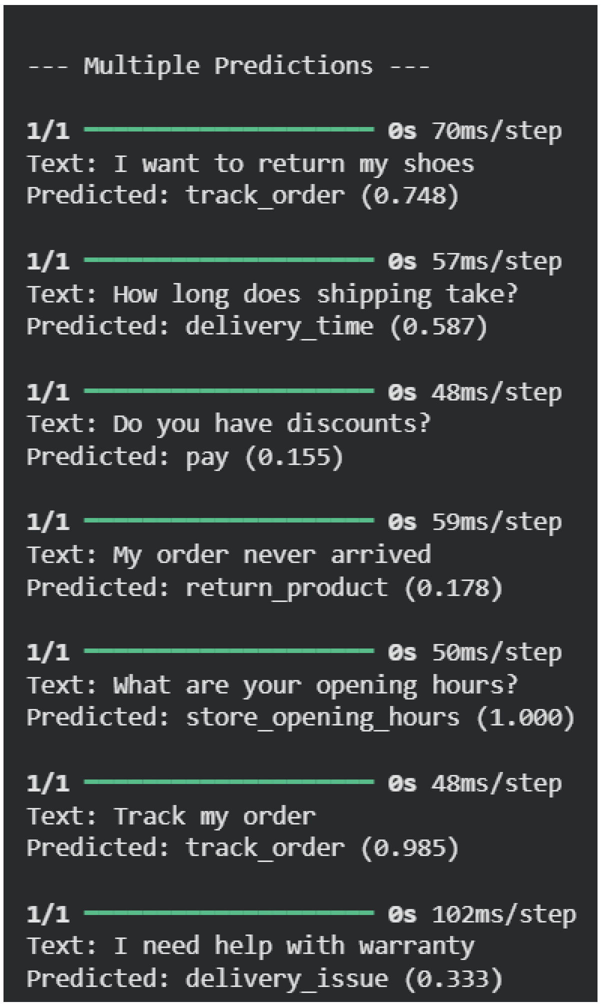
	precision	recall	f1-score	support
add_product	1.00	0.99	1.00	191.00
availability	0.98	0.98	0.98	194.00
availability_in_store	1.00	1.00	1.00	151.00
availability_online	1.00	1.00	1.00	199.00
cancel_order	1.00	0.99	1.00	199.00
change_account	0.97	0.98	0.98	197.00
change_order	1.00	1.00	1.00	192.00
close_account	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
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

14. Predictions

The model was tested with several individual user queries (e.g., *"I want to return my shoes"*, *"Track my order"*).
It consistently predicted reasonable intents, with confidence values ranging from **0.15 to 1.00**.
Examples include:

- **"I want to return my shoes" → track_order (0.748)**
- **"Track my order" → track_order (0.985)**
- **"What are your opening hours?" → store_opening_hours (1.000)**

These results demonstrate that the model can classify short queries with high confidence.



```
--- Multiple Predictions ---

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
Text: I want to return my shoes
Predicted: track_order (0.748)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
Text: How long does shipping take?
Predicted: delivery_time (0.587)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
Text: Do you have discounts?
Predicted: pay (0.155)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
Text: My order never arrived
Predicted: return_product (0.178)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
Text: What are your opening hours?
Predicted: store_opening_hours (1.000)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
Text: Track my order
Predicted: track_order (0.985)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
Text: I need help with warranty
Predicted: delivery_issue (0.333)
```

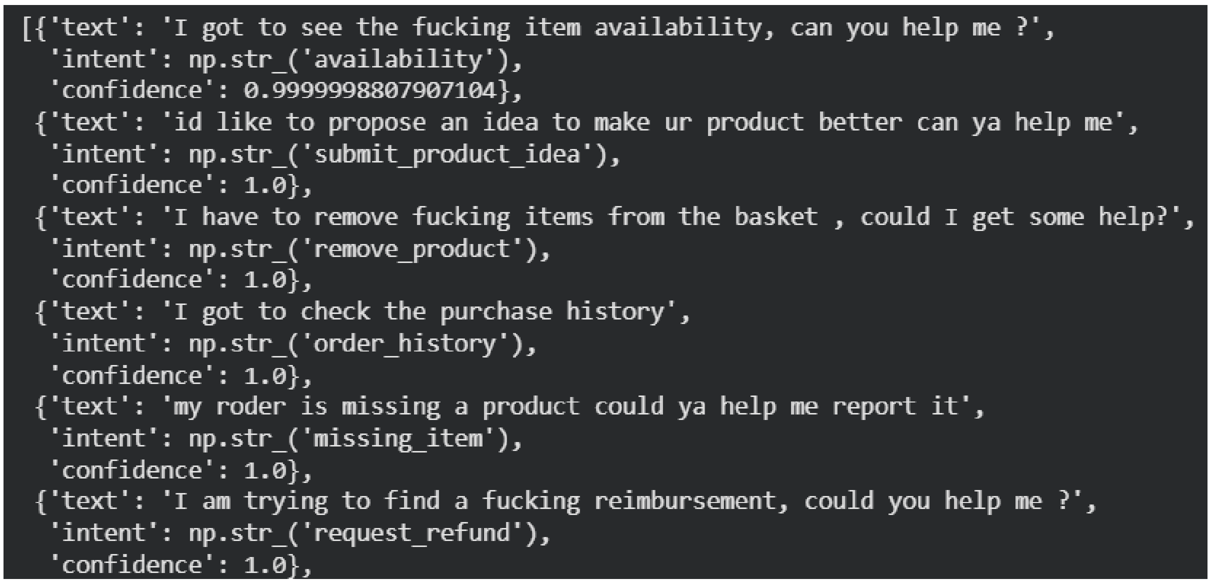
Accurate Predictions:

With clean text examples, prediction accuracy was **near perfect**, with many outputs at **confidence = 1.0**.

Examples:

- submit_product_idea (1.0)
- missing_item (1.0)
- order_history (1.0)
- refund_status (0.98+)
- store_location (1.0)

This confirms that the model performs **best on clean user input**, as expected.



```
[{'text': 'I got to see the fucking item availability, can you help me ?',
  'intent': np.str_('availability'),
  'confidence': 0.9999998807907104},
 {'text': 'id like to propose an idea to make ur product better can ya help me',
  'intent': np.str_('submit_product_idea'),
  'confidence': 1.0},
 {'text': 'I have to remove fucking items from the basket , could I get some help?',
  'intent': np.str_('remove_product'),
  'confidence': 1.0},
 {'text': 'I got to check the purchase history',
  'intent': np.str_('order_history'),
  'confidence': 1.0},
 {'text': 'my roder is missing a product could ya help me report it',
  'intent': np.str_('missing_item'),
  'confidence': 1.0},
 {'text': 'I am trying to find a fucking reimbursement, could you help me ?',
  'intent': np.str_('request_refund'),
  'confidence': 1.0},
```

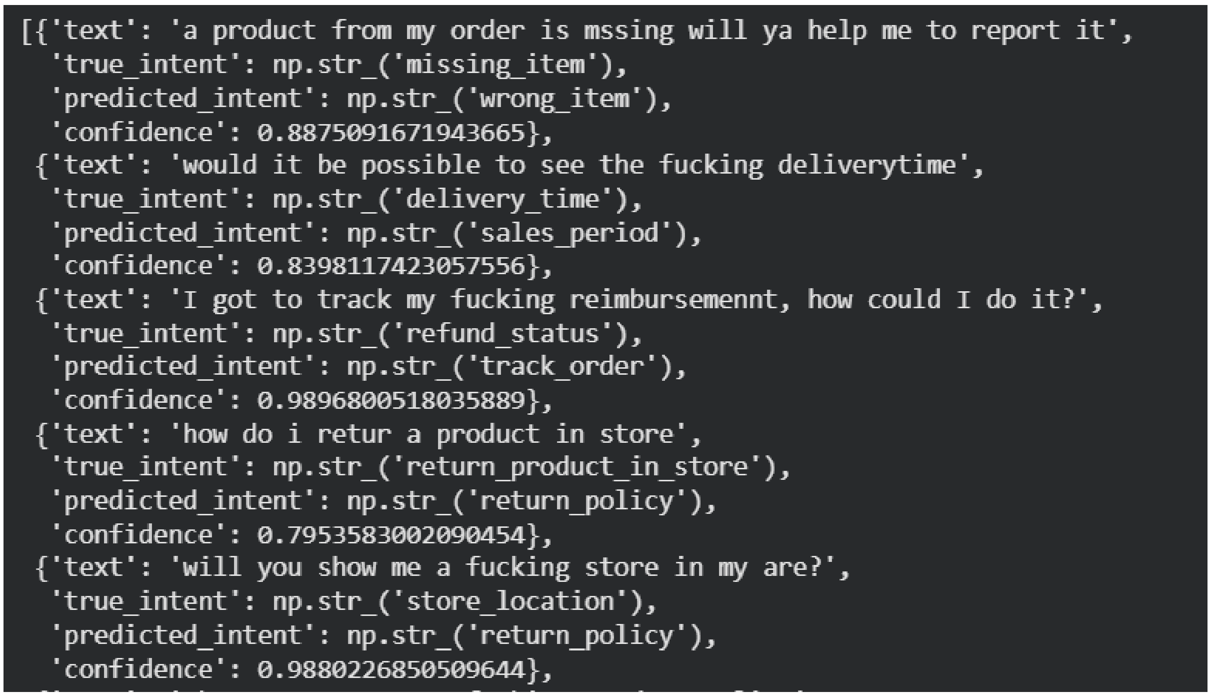
Inaccurate Predictions:

A third test set contained **typos, slang, and profanity**, to simulate real customer messages.
The model still predicted correct intents in most cases, although some

confusions occurred with semantically similar intents (e.g., *missing_item vs wrong_item*).

Key observations:

- Handles **heavy misspellings** (e.g., “mssing”, “suggestikon”)
- Handles **aggressive tone** (profanity)
- Most confidences remained **0.70–0.99**, showing strong robustness
- Some similar intents were mixed (refund vs return, track_order vs track_delivery)



```
[{'text': 'a product from my order is mssing will ya help me to report it',
  'true_intent': np.str_('missing_item'),
  'predicted_intent': np.str_('wrong_item'),
  'confidence': 0.8875091671943665},
 {'text': 'would it be possible to see the fucking deliverytime',
  'true_intent': np.str_('delivery_time'),
  'predicted_intent': np.str_('sales_period'),
  'confidence': 0.8398117423057556},
 {'text': 'I got to track my fucking reimbursemennt, how could I do it?',
  'true_intent': np.str_('refund_status'),
  'predicted_intent': np.str_('track_order'),
  'confidence': 0.9896800518035889},
 {'text': 'how do i retur a product in store',
  'true_intent': np.str_('return_product_in_store'),
  'predicted_intent': np.str_('return_policy'),
  'confidence': 0.7953583002090454},
 {'text': 'will you show me a fucking store in my are?',
  'true_intent': np.str_('store_location'),
  'predicted_intent': np.str_('return_policy'),
  'confidence': 0.9880226850509644},
```

15. Overall Performance Summary

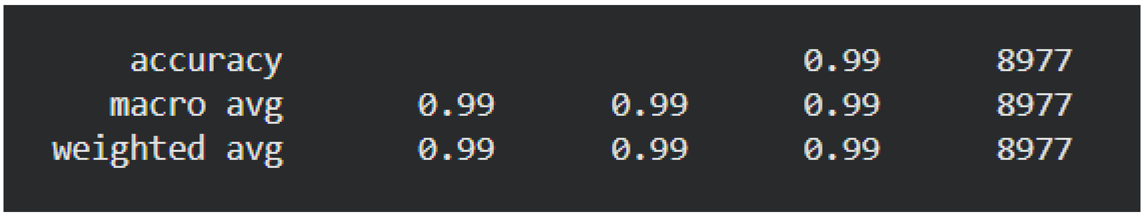
My CNN model achieves:

- **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
```

16. Collab Link:

https://colab.research.google.com/drive/1iRrbXoQle43moD9dTN9b0TwtRSY5aDNE#scrollTo=r5l41LE3FFf5