Laundromat Chatbot — BERT + Learnable (Full notebook walkthrough)

A friendly chatbot for your laundry shop that uses BERT embeddings to answer customer questions and can learn new Q&A while running. Saved knowledge persists in qa_data.json.

Install required packages

- **transformers:** provides BERT tokenizer & model.
- **torch:** PyTorch — backend to run BERT.
- **scikit-learn:** we use cosine_similarity to compare sentence vectors.

```
!pip install --upgrade pip
!pip install transformers torch scikit-learn --quiet
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.12/dist-packages (24.1.2)
Collecting pip
  Downloading pip-25.2-py3-none-any.whl.metadata (4.7 kB)
Downloading pip-25.2-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 69.6 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.1.2
    Uninstalling pip-24.1.2:
      Successfully uninstalled pip-24.1.2
Successfully installed pip-25.2
```

Import libraries

- **BertTokenizer:** converts text to tokens BERT understands.
- **BertModel:** pretrained BERT model that outputs embeddings.
- **torch:** runs model computations.
- **numpy:** numerical arrays and operations.
- **cosine_similarity:** gives similarity score between vectors.
- **json, os:** save/load knowledge to/from disk.
- **time:** tiny helper for timestamps (optional).

```
from transformers import BertTokenizer, BertModel
import torch
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import json
import os
import time
```

Load the BERT tokenizer & model (inference mode)

- We use "bert-base-uncased" — a common pretrained model.
- .to(device) moves model to GPU (if available) for speed.
- model.eval() sets the model for inference (turns off dropout etc.).

```
print("Loading BERT model (this may take ~30-90 seconds on first run)...")
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()
print("BERT loaded. Device:", device)
```

```
Loading BERT model (this may take ~30-90 seconds on first run)...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%                                  48.0/48.0 [00:00<00:00, 3.01kB/s]
vocab.txt: 100%                                              232k/232k [00:00<00:00, 11.4MB/s]
tokenizer.json: 100%                                         466k/466k [00:00<00:00, 45.1MB/s]
config.json: 100%                                            570/570 [00:00<00:00, 72.5kB/s]
model.safetensors: 100%                                      440M/440M [00:06<00:00, 52.3MB/s]
BERT loaded. Device: cuda
```

Helper: get BERT sentence embedding (mean pooling)

- Tokenize the text, run through BERT, get token vectors, and average them to get a single vector that represents the whole sentence.
- This is called mean pooling; simple and effective for small projects.

```
def get_embedding(text: str):
    """
    Convert text -> tokens -> BERT -> mean-pooled embedding (numpy array).
    Returns a 1D numpy array of shape (hidden_size,).
    """
    inputs = tokenizer(
        text,
        return_tensors="pt",
        truncation=True,
        padding=True,
        max_length=128
    )
    # move inputs to the model device
    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    # outputs.last_hidden_state shape: (batch_size=1, seq_len, hidden_dim)
    emb = outputs.last_hidden_state.mean(dim=1).cpu().numpy()[0]
    return emb
```

File helpers: save/load Q&A (persistence)

- qa_data.json stores the knowledge base.
- This makes the bot remember what it learned after the notebook closes.

```
QA_FILE = "qa_data.json"

def load_qa(path=QA_FILE):
    """Load Q&A from JSON; create initial file if not found."""
    if os.path.exists(path):
        with open(path, "r", encoding="utf-8") as f:
            return json.load(f)
    else:
        # if not exists, create with INITIAL_QA (defined below)
        with open(path, "w", encoding="utf-8") as f:
            json.dump(INITIAL_QA, f, ensure_ascii=False, indent=2)
        return INITIAL_QA.copy()

def save_qa(qa_dict, path=QA_FILE):
    """Save Q&A dictionary to disk."""
    with open(path, "w", encoding="utf-8") as f:
        json.dump(qa_dict, f, ensure_ascii=False, indent=2)
```

Define initial laundry-shop Q&A (lots of realistic questions)

- These are realistic FAQs for a laundry shop (Suds & Shine).
- They give the bot a strong starting knowledge base.

```
INITIAL_QA = {
    "Hi": "Hello! Welcome to Suds & Shine Laundry 🧼. How can I help you today?",
    "Hello": "Hi there! Welcome to Suds & Shine Laundry. How may I assist?",
    "What services do you offer?": "We offer wash & fold, dry cleaning, ironing, stain removal, sofa and curtain cleaning, an
    "What are your working hours?": "We're open Monday–Saturday, 8:00 AM to 8:00 PM. Closed on Sundays.",
    "Where are you located?": "We're at 123 Clean Street, Near Karve Nagar, Pune.",
    "How much does a basic wash cost?": "A basic wash & fold starts at ₹50 per kg.",
    "How much for dry cleaning a shirt?": "Dry cleaning for a shirt is ₹80.",
    "Do you do same-day service?": "Yes — for small loads, we offer same-day service if dropped off before 10:00 AM (subject
    "Do you offer pickup and delivery?": "Yes! Free pickup & delivery within a 5 km radius. Charges apply beyond that.",
    "How long does it take?": "Turnaround is usually 24–48 hours for regular service; express options are available.",
    "Can you remove tough stains?": "We handle many tough stains. Please point them out during drop-off for the best treatmen
    "Do you wash delicate fabrics?": "Yes — we have special care processes for silk, wool, and other delicates.",
    "Do you accept online payments?": "We accept cash, UPI, and card payments on delivery.",
    "Do you have membership plans?": "Yes — monthly packages start at ₹999 and include discounts on every order.",
    "How do I track my order?": "You can track orders via SMS/WhatsApp updates. Share your order ID with us.",
    "What is your contact number?": "Call or WhatsApp us at +91 98765 43210.",
    "Do you provide garment repairs?": "Minor repairs like small stitches and buttons are available at an extra charge.",
    "Are your detergents eco-friendly?": "We offer eco-friendly cleaning on request (may be an extra charge).",
    "Can I schedule a regular pickup?": "Yes — weekly or bi-weekly pickups can be scheduled.",
    "Do you offer wedding dress cleaning?": "Yes — we offer specialist cleaning for wedding gowns (appointment required).",
    "What is your cancellation policy?": "You can cancel within 1 hour of booking without charge; after that, a small fee app
}
```

Load / initialize Q&A and compute embeddings for stored questions

- We compute embeddings for each stored question once so matching is faster later.
- If you add new Q&A later, we'll compute its embedding then.

```
# Initialize Q&A file if missing, then load
if not os.path.exists(QA_FILE):
    print("Creating initial qa_data.json with default laundry Q&A...")
qa = load_qa(QA_FILE)

# Build embeddings dict for the stored questions
print("Computing embeddings for stored questions (this may take a few seconds)...")
embeddings = {}
for q in qa.keys():
    embeddings[q] = get_embedding(q)
print("Done. Stored Q&A count:", len(qa))
```

```
Creating initial qa_data.json with default laundry Q&A...
Computing embeddings for stored questions (this may take a few seconds)...
Done. Stored Q&A count: 21
```

Chatbot matching logic (threshold-based)

- The bot computes similarity of the user's input to each stored question and picks the highest.
- threshold defines how confident the bot must be to give a stored answer. Lower = more permissive, Higher = stricter.

```
def get_bot_response(user_text, qa_dict, emb_cache, threshold=0.55):
    """
    Return (answer or None, matched_question, score).
    If answer is None -> bot is not confident.
    """
    user_emb = get_embedding(user_text).reshape(1, -1)  # shape (1, hidden_dim)
    questions = list(emb_cache.keys())
    emb_matrix = np.vstack([emb_cache[q] for q in questions])  # shape (n_questions, hidden_dim)
    sims = cosine_similarity(user_emb, emb_matrix)[0]  # shape (n_questions,)
    best_idx = int(np.argmax(sims))
    best_score = float(sims[best_idx])
    best_q = questions[best_idx]
    if best_score >= threshold:
        return qa_dict[best_q], best_q, best_score
    else:
        # not confident enough
        return None, best_q, best_score
```

Interactive notebook chat loop (teach & save)

- If the bot is uncertain, it asks you to teach the correct answer.
- New Q&A are saved to disk immediately (so they persist across notebook restarts).

```
print("🫧 Suds & Shine — BERT Learnable Chatbot (type 'quit' to exit)\n")

while True:
    user = input("You: ").strip()
    if not user:
        continue
    if user.lower() in ("quit", "exit"):
        print("Goodbye! 👋")
        break

    answer, matched_q, score = get_bot_response(user, qa, embeddings, threshold=0.55)

    if answer:
        print(f"Bot: {answer}  (matched: '{matched_q}' | score: {score:.3f})\n")
    else:
        print(f"Bot: I don't know the answer yet. Closest stored question: '{matched_q}' (score {score:.3f})")
        teach = input("Would you like to teach me the correct answer? (type answer or leave blank to skip): ").strip()
        if teach:
            # store and compute embedding
            qa[user] = teach
            embeddings[user] = get_embedding(user)
            save_qa(qa, QA_FILE)
            print("Bot: Thanks — I've learned that! ✅\n")
        else:
            print("Bot: No problem — you can teach me anytime.\n")
```

Quick programmatic tests (example queries)

- Use this to quickly check multiple queries and see if the current knowledge covers them.

```
tests = [
    "Do you do curtain cleaning?",
    "How much for a shirt dry clean?",
    "Can I schedule weekly pickup?",
    "What time do you open on Saturdays?"
]

for t in tests:
    ans, matched, sc = get_bot_response(t, qa, embeddings)
    if ans:
        print(f"Q: {t}\nA: {ans} (matched '{matched}' | {sc:.3f})\n")
    else:
        print(f"Q: {t}\nA: I don't know yet. Closest match: '{matched}' | {sc:.3f}\n")
```

```
Q: Do you do curtain cleaning?
A: Minor repairs like small stitches and buttons are available at an extra charge. (matched 'Do you provide garment repairs?'

Q: How much for a shirt dry clean?
A: Dry cleaning for a shirt is ₹80. (matched 'How much for dry cleaning a shirt?' | 0.931)

Q: Can I schedule weekly pickup?
A: Yes — weekly or bi-weekly pickups can be scheduled. (matched 'Can I schedule a regular pickup?' | 0.969)

Q: What time do you open on Saturdays?
A: Yes — for small loads, we offer same-day service if dropped off before 10:00 AM (subject to availability). (matched 'Do you
```

Inspect & manage saved knowledge from the notebook

- Use this to check what the bot currently knows.
- To reset to defaults, remove qa_data.json (uncomment lines to execute).

```
# View saved Q&A count and sample
print("Total Q&A saved:", len(qa))
for i, (q, a) in enumerate(qa.items()):
    print(f"{i+1}. Q: {q}\n   A: {a}\n")
    if i >= 20:
        break  # show first 20 only

# To reset knowledge (careful): uncomment the following two lines
```

```
# os.remove(QA_FILE)
# print("Deleted qa_data.json - restart the notebook to recreate with INITIAL_QA.")
```

```
Total Q&A saved: 21
1. Q: Hi
   A: Hello! Welcome to Suds & Shine Laundry 🧼. How can I help you today?

2. Q: Hello
   A: Hi there! Welcome to Suds & Shine Laundry. How may I assist?

3. Q: What services do you offer?
   A: We offer wash & fold, dry cleaning, ironing, stain removal, sofa and curtain cleaning, and pickup & delivery.

4. Q: What are your working hours?
   A: We're open Monday–Saturday, 8:00 AM to 8:00 PM. Closed on Sundays.

5. Q: Where are you located?
   A: We're at 123 Clean Street, Near Karve Nagar, Pune.

6. Q: How much does a basic wash cost?
   A: A basic wash & fold starts at ₹50 per kg.

7. Q: How much for dry cleaning a shirt?
   A: Dry cleaning for a shirt is ₹80.

8. Q: Do you do same-day service?
   A: Yes — for small loads, we offer same-day service if dropped off before 10:00 AM (subject to availability).

9. Q: Do you offer pickup and delivery?
   A: Yes! Free pickup & delivery within a 5 km radius. Charges apply beyond that.

10. Q: How long does it take?
   A: Turnaround is usually 24–48 hours for regular service; express options are available.

11. Q: Can you remove tough stains?
   A: We handle many tough stains. Please point them out during drop-off for the best treatment.

12. Q: Do you wash delicate fabrics?
   A: Yes — we have special care processes for silk, wool, and other delicates.

13. Q: Do you accept online payments?
   A: We accept cash, UPI, and card payments on delivery.

14. Q: Do you have membership plans?
   A: Yes — monthly packages start at ₹999 and include discounts on every order.

15. Q: How do I track my order?
   A: You can track orders via SMS/WhatsApp updates. Share your order ID with us.

16. Q: What is your contact number?
   A: Call or WhatsApp us at +91 98765 43210.

17. Q: Do you provide garment repairs?
   A: Minor repairs like small stitches and buttons are available at an extra charge.

18. Q: Are your detergents eco-friendly?
   A: We offer eco-friendly cleaning on request (may be an extra charge).

19. Q: Can I schedule a regular pickup?
   A: Yes — weekly or bi-weekly pickups can be scheduled.
```

Tips & next steps (recommended improvements)

- Use sentence-transformers (all-miniLM) for faster & better sentence similarity in production.
- **Add context:** pass previous user/Bot messages concatenated to the input to handle follow-ups.
- **Build a Streamlit UI:** nicer interface for customers (I'll include a starter app.py section below).
- **Rate limits & batching:** if your bot gets heavy traffic, precompute embeddings and use a more efficient nearest-neighbor index (FAISS).
- Backup qa_data.json periodically (versioning with timestamps).

```
Start coding or generate with AI.
```