رواد مصر الرقمية

DKH3_AIS3_S1
June-December 2025

Final Report
“Chatbot”

1- Data Collection and Preprocessing:

1. Dataset Loading:

The code uses the Persona-Chat dataset from Hugging Face,

dataset = load_dataset("Cynaptics/persona-chat")

This dataset contains dialogues between two speakers (Persona A and Persona B) and is commonly used for building conversational AI systems.

2. Dataset Splitting

To ensure robust evaluation and avoid overfitting, the dataset is randomly split into:

- 80% Train
- 10% Test
- 10% Validation

3. Text Cleaning

A custom function clean_text() performs basic cleaning:

1. Converts all text to lowercase
2. Removes URLs
3. Removes extra whitespace
4. Ensures non-string values are handled safely

This step improves model consistency and reduces noise during training.

4. Adding Special Tokens

The function add_**special_tokens()** standardizes speaker identities:
"Persona A" → <usr>
"Persona B" → <bot>
This helps the GPT-2 model understand who is speaking and mimic realistic conversational turns.

5. Dialogue to Training Pairs

The function dialog_to_pairs() transforms multi-turn dialogue into paired training examples.
Each pair follows the template:
<usr> USER_UTTERANCE <bot> BOT_RESPONSE <|endoftext|>
Where <|endoftext|> signals the end of each training sample.
This creates high-quality input–output sequences suitable for causal language modeling, where GPT-2 predicts the next tokens in a conversation.

6. Dataset Flattening

After mapping the dialogues into pairs, the dataset contains nested lists
This ensures the dataset is linear and ready for tokenization.

7. Tokenization using GPT-2 Tokenizer

The GPT-2 tokenizer is loaded:

tokenizer = AutoTokenizer.from_pretrained("gpt2")

Since GPT-2 has no pad token, the code sets:

tokenizer.pad_token = tokenizer.eos_token

Padding is then applied:

- max_length = 128
- Truncation enabled
- Padded sequences aligned for batch training

This produces input_ids, attention_mask, etc.

8. Label Creation

For causal language modeling, the labels are simply a copy of the input:

labels = input_ids

This enables the model to learn next-token prediction.

9. Saving the Final Dataset

The fully processed dataset is saved locally

tokenized_datasets.save_to_disk("./processed_dataset")

This makes it directly usable by trainers like:

- HuggingFace Trainer
- PyTorch DataLoader
- LoRA fine-tuning scripts

2- Model Development

This section describes the model training pipeline used to fine-tune a GPT-2 language model on the processed Persona-Chat dataset.

- It is lightweight and fast to train
- It supports conversational fine-tuning
- It works well with special tokens and dialogue structures

1. Loading the Preprocessed Dataset

loading the dataset that was previously saved during the preprocessing stage:

datasets = load_from_disk(DATA_PATH)

- This dataset contains:
- Tokenized input sequences (input_ids)
- Corresponding labels for causal language modeling
- Train, validation, and test splits

2. Initializing the Tokenizer and Base Model

The GPT-2 tokenizer and model are loaded:

tokenizer = AutoTokenizer.from_pretrained("gpt2")

model = AutoModelForCausalLM.from_pretrained("gpt2")

Padding Configuration

Since GPT-2 has no native padding token, the script sets:

tokenizer.pad_token = tokenizer.eos_token

This ensures that all sequences have equal length during training by using the end-of-text token as padding.

3. Defining Training Configuration

Training parameters are defined using Hugging Face's TrainingArguments:

training_args = TrainingArguments(...)

Key configurations include:

num_train_epochs = 6

The model iterates over the entire training set six times.

per_device_train_batch_size = 4

Small batch size ensures training compatibility with limited memory environments.

Evaluation Strategy

evaluation_strategy = "epoch"

The model is evaluated on the validation set after each epoch.

Model Checkpoints

save_strategy = "epoch"

load_best_model_at_end = True

The script saves the model each epoch and restores the best performing version after training.

GPU Acceleration

fp16 = torch.cuda.is_available()

Half-precision training is used automatically when a GPU is available, resulting in faster and more efficient training.

4. Trainer Setup

The Hugging Face Trainer class manages the training loop

The trainer handles:

- Data batching
- Forward and backward passes
- Optimization
- Evaluation
- Logging
- Checkpointing

5. Running the Training Loop

During this stage:

The model learns to predict the next token in a sequence (causal language modeling)

Loss decreases over time

Evaluation metrics are computed after each epoch

Best model weights are tracked

6. Training Outputs

A fully fine-tuned conversational GPT-2 model

Model + tokenizer saved to:

./gpt2_chatbot_model/

Logs and checkpoints stored for reproducibility

---

3. Advanced Techniques

1. Use of Special Tokens

The dialogue structure **<usr>, <bot>**, and **<|endoftext|>** improves:

Context recognition

Turn-taking behavior

Response boundaries

This significantly increases conversational consistency.

2. Attention Mechanisms (GPT-2 Self-Attention)

GPT-2 uses multi-head self-attention, which allows the model to:

Identify long-range dependencies in conversation

Understand context from previous conversation turns

Maintain persona consistency

Although attention is built into GPT-2, our preprocessing ensures that the attention mechanism receives well-structured inputs.

3. Pipeline Integration

The project follows a multi-step AI pipeline:

Data Preprocessing

Model Training

Evaluation Metrics Calculation

Deployment + Continuous Learning

This pipeline provides a clear MLOps-friendly structure.

4. Evaluation Metrics

The evaluation script computed:

Perplexity (PPL)

BLEU score

ROUGE-1 / ROUGE-2 / ROUGE-L

BERTScore (Precision, Recall, F1)

---

4. Frontend Implementation

The frontend of the chatbot application is implemented using HTML, CSS, and JavaScript to provide an interactive and responsive user interface. The main functionalities are as follows:

- **Message Handling:**

Users can type messages in a textarea input and send them either by clicking the "Send" button or pressing the Enter key.

Each message is displayed in the chat window with a timestamp and a user/bot avatar.

The chat dynamically scrolls to show the latest messages.

- **Typing Indicator:**

When a message is sent, a typing indicator appears to simulate the bot "thinking" while the response is being generated.

The indicator is removed once the response is received from the backend.

- **Responsive Input Field:**

The height of the input textarea adjusts automatically based on the content, up to a maximum height, to ensure a clean and usable layout.

- **Send Button Activation:**

The "Send" button is enabled only when the input field contains text, preventing empty messages from being sent.

- **Theme Toggle:**

The interface supports light and dark themes, which can be toggled by the user.

The selected theme is saved in localStorage to persist between sessions.

The application also respects the user's system theme preference on first load.

- **Integration with Backend:**

Messages are sent to the FastAPI backend (/chat endpoint) via a POST request.

The bot response received is displayed in the chat interface, ensuring a real-time conversational experience.