「加油站POS系統及周邊設備問題處理」資料庫概述

及待解決問題

1. **提案目的：**當加油站人員在操作POS系統時遇到問題，通常會在Line群組中發問 。為了避免重複提問和知識流失，我們將已累積的問題與解決方法進行統整，讓人員能先參考過往經驗自行排除簡單問題，同時也作為後續系統改善的參考依據 。
2. **系統架構簡述**

Line機器人系統採用GCP serverless架構（Cloud Run），方便監控效益且易於移植 。其核心運作流程簡述如下：

* 1. **使用者輸入**：加油站同仁透過Line官方帳號輸入問題或點擊表單 。
  2. **Web App處理**：Line Webhook接收請求後傳至後端Web App 。
  3. **問題比對**：Web App從**知識資料庫（目前為Google Sheet）** 取得問答資料，並同時運用
  4. **BM25（字面匹配）** 和 **語意比對（Semantic Similarity，透過Embedding和FAISS）** 技術，計算使用者問題與資料庫中問題的相關性綜合得分 。
  5. **AI生成回答**：根據綜合得分，找到最相似的答案，並透過AI模型（如Gemini）生成自然語言回答 。
  6. **回饋記錄**：所有使用者對話及回饋（讚/倒讚）都會被記錄在 **Firestore（紀錄層）** ，作為後續模型優化的依據 。

1. **待解決事項**
   1. 資料庫內容雜亂，敘述不一，多無用字符，待統整，讓機器人好辨識。

* **問題現狀**：目前知識資料庫（Google Sheet）中的問答內容，由於來源廣泛且未經嚴格規範，存在格式不統一、敘述混亂、夾雜多餘符號或非必要資訊等問題 。這嚴重影響了AI模型的判讀效率與準確性，導致機器人無法有效匹配問題或提供精確答案 。
* **解決目標**：您需要對現有資料庫內容進行全面審閱和清洗。這包括：
  + 建立並遵循一套統一的問答格式規範（例如：移除特殊符號、統一標點符號、簡潔問題描述）。
  + 清洗或刪除無效、重複或過時的問答內容。
  + 確保問題描述與解決方案之間邏輯清晰，易於AI理解。
  1. 新增100題問題。
* **問題現狀**：專案測試結果顯示，目前資料庫內容仍不完整，尚無法精確回應所有問題 。AI模型的表現與回答準確性高度依賴於訓練資料的數量與品質 。
* **解決目標**：您需要在現有基礎上，至少新增 **100 筆**高品質的問題與解決方案。這要求您：
  + 主動與加油站站長、值班站長及資深人員溝通，收集高頻率出現但資料庫中缺乏的POS系統及周邊設備問題。
  + 將收集到的問題和解決方案，按照上述第一點的統一格式，仔細整理並輸入到Google Sheet資料庫中。
  + 確保新增的問題具有代表性，能涵蓋日常操作中常見的困擾。
  1. **需優化以下10題的回答精準度：**
* 這10個問題代表了使用者目前查詢時，機器人回答精準度較差的典型案例。您的目標是針對這10個具體問題，分析現有資料庫和機器人回答流程，找出導致不精準的原因，並提出並實施優化方案，最終使機器人能夠給出精準且實用的回答。
* **待優化問題列表：** a. 農機可以使用的支付方式 b. 捷利卡無法使用 c. 悠遊付、全支付退款方式? d. 台糖禮品卷怎麼用 e. 9大條碼支付(「LINE Pay」(含LINE Pay Money)、「一卡通MONEY」、「Pi 拍錢包」、「街口」、「歐付寶」、 「橘子支付」、「車麻吉」、「中油PAY」、「台灣Pay」)退款方式? f. 中油pay 如何作廢 g. 中油要上傳了，出現『班報尚未完全產生，無法建立日報』，怎麼會這樣? h. 台糖特約車牌申請 i. 人拿模糊中獎發票需要補開 j. 可以一筆交易混合付款別嗎

//主要程式碼  
import os

import json

import time

import threading

from datetime import datetime

# Flask and LINE Bot imports

from flask import Flask, abort, request

from linebot import LineBotApi, WebhookHandler

from linebot.exceptions import InvalidSignatureError, LineBotApiError

from linebot.models import \*

# Google services imports

import pygsheets

import google.generativeai as genai

from google.cloud import firestore

from google.oauth2 import service\_account

# ML and NLP imports

import numpy as np

from rank\_bm25 import BM25Okapi

import jieba

# Time zone

import pytz

###############################################################################

# CONFIGURATION AND INITIALIZATION

###############################################################################

# 設定版本代碼和時區

VERSION\_CODE = "09.06.2025"

GMT\_8 = pytz.timezone("Asia/Taipei")

print(f"Starting application - Version Code: {VERSION\_CODE}")

app = Flask(\_\_name\_\_)

# Initialize Gemini API

gemini\_api\_key = os.getenv("GEMINI\_API\_KEY")

genai.configure(api\_key=gemini\_api\_key)

generation\_model = genai.GenerativeModel("gemini-2.0-flash")

# LINE Bot setup

line\_bot\_api = LineBotApi(os.environ.get("LINE\_BOT\_CHANNEL\_ACCESS\_TOKEN"))

handler = WebhookHandler(os.environ.get("LINE\_BOT\_CHANNEL\_SECRET"))

ALLOWED\_DESTINATION = os.environ.get("ALLOWED\_DESTINATION")

# Google Sheets setup

gc = pygsheets.authorize(service\_account\_file='service\_account\_key.json')

sheet = gc.open\_by\_url(os.environ.get("GOOGLESHEET\_URL"))

# Firestore setup

def get\_firestore\_client\_from\_env():

firestore\_json = os.getenv("FIRESTORE")

if not firestore\_json:

raise ValueError("FIRESTORE environment variable is not set.")

cred\_info = json.loads(firestore\_json)

credentials = service\_account.Credentials.from\_service\_account\_info(cred\_info)

return firestore.Client(credentials=credentials, project=cred\_info["project\_id"])

db = get\_firestore\_client\_from\_env()

###############################################################################

# DATA LOADING AND PREPROCESSING

###############################################################################

# Load questions and answers from Google Sheets 主要QA

def load\_sheet\_data():

# Main questions

main\_ws = sheet.worksheet("title", "表單回應")

main\_questions = main\_ws.get\_col(3, include\_tailing\_empty=False)

main\_answers = main\_ws.get\_col(4, include\_tailing\_empty=False)

# 取得 "CPC問題" 和 "CPC點數" 的值

cpc\_ws = sheet.worksheet("title", "中油點數")

cpc\_questions = cpc\_ws.get\_col(8, include\_tailing\_empty=False)

cpc\_answers = cpc\_ws.get\_col(9, include\_tailing\_empty=False)

cpc\_list = cpc\_ws.get\_col(1, include\_tailing\_empty=False)

return main\_questions + cpc\_questions, main\_answers + cpc\_answers, cpc\_list

questions\_in\_sheet, answers\_in\_sheet, cpc\_list = load\_sheet\_data()

# Load synonyms dictionary

def load\_synonyms():

syn\_ws = sheet.worksheet("title", "同義詞")

synonym\_rows = syn\_ws.get\_all\_values()

synonym\_dict = {}

for row in synonym\_rows:

synonyms = [word.strip() for word in row if word.strip()]

for word in synonyms:

synonym\_dict[word] = set(synonyms) - {word}

return synonym\_dict

synonym\_dict = load\_synonyms()

# Initialize ML models

# 先對問句進行分詞

tokenized\_questions = [list(jieba.cut(q)) for q in questions\_in\_sheet]

# 建立 BM25 模型

bm25 = BM25Okapi(tokenized\_questions)

# 載入中文句向量模型

\_model = None

def get\_model():

global \_model

if \_model is None:

from sentence\_transformers import SentenceTransformer

\_model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")

return \_model

question\_embeddings = get\_model().encode(questions\_in\_sheet)

###############################################################################

# SEARCH AND RETRIEVAL FUNCTIONS

###############################################################################

def expand\_query(query):

"""擴展查詢詞，加入同義詞"""

words = jieba.lcut(query)

expanded\_words = set(words)

for word in words:

if word in synonym\_dict:

expanded\_words.update(synonym\_dict[word])

return " ".join(expanded\_words)

def retrieve\_top\_n(query, n=2, threshold=5, high\_threshold=10):

"""取得最相似的問題

##作法

1.使用Sentence Transformers進行相似度計算

2.使用BM25強化搜索

3.閥值為5，超過才列為答案

4.最多選擇2個答案

"""

try:

expanded\_query = expand\_query(query)

tokenized\_query = list(jieba.cut(expanded\_query))

# BM25 排序

bm25\_scores = bm25.get\_scores(tokenized\_query)

# Sentence Transformers 相似度計算(餘弦相似度)

query\_embedding = get\_model().encode([query])[0]

semantic\_scores = np.dot(question\_embeddings, query\_embedding)

# 兩者加權平均（可調整權重）

combined\_scores = 0.7 \* np.array(bm25\_scores) + 0.3 \* semantic\_scores

# 1. 篩選出超過基本閾值的結果

above\_threshold\_indices = [

i for i, score in enumerate(combined\_scores) if score >= threshold

]

if not above\_threshold\_indices:

return []

# 2. 按照綜合分數排序

sorted\_indices = sorted(

above\_threshold\_indices, key=lambda i: combined\_scores[i], reverse=True

)

high\_score\_indices = [

i for i in sorted\_indices if combined\_scores[i] >= high\_threshold

]

result = []

if len(high\_score\_indices) >= 2:

# 如果有兩個或以上高分結果，返回前n個

result = [

{

"question": questions\_in\_sheet[i],

"answer": answers\_in\_sheet[i],

"bm25\_score": float(bm25\_scores[i]),

"semantic\_score": float(semantic\_scores[i]),

"combined\_score": float(combined\_scores[i]),

}

for i in high\_score\_indices[:n]

]

threading.Thread(

target=record\_question\_for\_answer,

args=(questions\_in\_sheet[high\_score\_indices[0]],),

).start()

else:

# 如果沒有或只有一個高分結果，只返回最高分的一個

i = sorted\_indices[0]

result = [

{

"question": questions\_in\_sheet[i],

"answer": answers\_in\_sheet[i],

"bm25\_score": float(bm25\_scores[i]),

"semantic\_score": float(semantic\_scores[i]),

"combined\_score": float(combined\_scores[i]),

}

]

threading.Thread(

target=record\_question\_for\_answer,

args=(questions\_in\_sheet[sorted\_indices[0]],),

).start()

return result

except Exception as e:

print(f"Error in retrieve\_top\_n: {str(e)}")

return []

###############################################################################

# LLM AND RESPONSE PROCESSING

###############################################################################

def reply\_by\_LLM(finalanswer, model):

"""使用LLM生成自然語言回覆"""

try:

prompt = f"""你是知識問答客服，請將{ finalanswer }直接轉成自然語言。

##條件

1.口氣禮貌親切簡潔，像是和使用者對話

2.若finalanswer為空[]，則回覆:此問題目前找不到合適解答，請聯絡積慧幫忙協助

3.若finalanswer不為空[]，最後請換行後加一句:若此答案無法解決您問題，請換個問題再問一次或是聯絡積慧幫忙協助

4.不要解釋以上回覆條件，直接回覆答案

5.不要反問使用者

"""

answer\_in\_human = model.generate\_content(prompt)

return answer\_in\_human

except Exception as e:

print(f"Error in reply\_by\_LLM: {str(e)}")

return None

def extract\_chinese\_results\_new(response):

"""從模型回應中提取中文內容"""

try:

text\_content = response.candidates[0].content.parts[0].text

if "\\u" in text\_content:

decoded\_text = text\_content.encode().decode("unicode\_escape")

return decoded\_text

return text\_content

except (AttributeError, IndexError, UnicodeError):

return ""

def find\_closest\_question\_and\_llm\_reply(query):

"""主要的問答處理函數"""

try:

top\_matches = retrieve\_top\_n(query)

if not top\_matches:

return {

"answer": "目前找不到合適的答案，請再試一次或換個問法",

"top\_matches": [],

}

answers\_only = [match["answer"] for match in top\_matches]

result = reply\_by\_LLM(answers\_only, generation\_model)

answer\_to\_line = extract\_chinese\_results\_new(result)

return {"answer": answer\_to\_line, "top\_matches": top\_matches}

except Exception as e:

print(f"Error in find\_closest\_question\_and\_llm\_reply: {str(e)}")

return {

"answer": "此問題目前找不到合適解答，請聯絡積慧幫忙協助",

"top\_matches": [],

}

###############################################################################

# DATA RETRIEVAL FUNCTIONS

###############################################################################

def get\_top\_questions():

"""獲取熱門問題前5名"""

try:

ranking\_ws = sheet.worksheet("title", "熱門排行")

print("Found '熱門排行' worksheet.")

except pygsheets.WorksheetNotFound:

print("熱門排行 worksheet not found.")

return []

top\_ranking\_records = ranking\_ws.get\_all\_records()[:5]

top\_questions = []

main\_ws = sheet.worksheet("title", "表單回應")

main\_records = main\_ws.get\_all\_records()

for record in top\_ranking\_records:

full\_question = next(

(item for item in main\_records if item["問題描述"] == record["項目"]), None

)

if full\_question:

top\_questions.append({

"排名": record["排名"],

"項目": record["項目"],

"問題描述": full\_question["問題描述"],

"解決方式": full\_question["解決方式"],

})

print(f"Top 5 questions with descriptions: {top\_questions}")

return top\_questions

def get\_unique\_categories():

"""獲取唯一問題分類"""

try:

main\_ws = sheet.worksheet("title", "表單回應")

categories\_column = main\_ws.get\_col(2)

unique\_categories = sorted(

list(set(cat.strip() for cat in categories\_column[1:] if cat.strip()))

)

print(f"Found {len(unique\_categories)} unique categories: {unique\_categories}")

return unique\_categories

except Exception as e:

print(f"Error in get\_unique\_categories: {str(e)}")

return []

def get\_questions\_by\_category(category):

"""根據分類獲取問題"""

try:

main\_ws = sheet.worksheet("title", "表單回應")

all\_data = main\_ws.get\_all\_values()

questions = []

for row in all\_data[1:]:

if len(row) > 2 and row[1].strip() == category.strip():

question\_text = row[2].strip()

if question\_text:

questions.append({

"問題描述": question\_text,

"解決方式": "",

})

print(f"Total {len(questions)} questions found for category '{category}'")

return questions

except Exception as e:

print(f"Error in get\_questions\_by\_category: {str(e)}")

return []

def find\_solution\_by\_click\_question(question\_text):

"""找對應問題的解決方式"""

try:

main\_ws = sheet.worksheet("title", "表單回應")

all\_data = main\_ws.get\_all\_values()

for row in all\_data[1:]:

if len(row) > 3 and row[2].strip() == question\_text.strip():

solution = row[3].strip()

print(f"Found solution for question '{question\_text}': {solution}")

return solution

print(f"No solution found for question '{question\_text}'")

return None

except Exception as e:

print(f"Error in find\_solution\_by\_question: {str(e)}")

return None

def get\_oil\_points\_column\_a():

"""獲取中油點數資料"""

if not cpc\_list or len(cpc\_list) == 0:

return "中油點數表單的 A 欄沒有資料。"

return "\n".join(cpc\_list)

###############################################################################

# LOGGING FUNCTIONS

###############################################################################

def record\_question(user\_id, user\_input):

"""記錄用戶問題到統計紀錄"""

gc = pygsheets.authorize(service\_account\_file='service\_account\_key.json')

sheet = gc.open\_by\_url(os.environ.get("GOOGLESHEET\_URL"))

try:

profile = line\_bot\_api.get\_profile(user\_id)

user\_name = profile.display\_name

print(f"Fetched user profile: {user\_name}")

except LineBotApiError as e:

user\_name = "Unknown"

print(f"Error getting user profile: {e}")

try:

stats\_ws = sheet.worksheet("title", "統計紀錄")

print("Found '統計紀錄' worksheet.")

except pygsheets.WorksheetNotFound:

stats\_ws = sheet.add\_worksheet("統計紀錄")

stats\_ws.update\_row(1, ["時間", "使用者ID", "使用者名稱", "詢問文字"])

print("Created '統計紀錄' worksheet.")

timestamp = datetime.now(GMT\_8).strftime("%Y-%m-%d %H:%M:%S")

record\_data = [timestamp, user\_id, user\_name, user\_input]

stats\_ws.insert\_rows(row=1, values=record\_data, inherit=True)

print(f"Recorded question: {record\_data}")

def record\_question\_for\_answer(question\_for\_answer):

"""記錄回答問題到回答工作表"""

gc = pygsheets.authorize(service\_account\_file='service\_account\_key.json')

sheet = gc.open\_by\_url(os.environ.get("GOOGLESHEET\_URL"))

try:

reply\_ws = sheet.worksheet("title", "回答")

print("Found '回答' worksheet.")

except pygsheets.WorksheetNotFound:

reply\_ws = sheet.add\_worksheet("回答")

reply\_ws.update\_row(1, ["時間", "問題"])

print("Created '回答' worksheet.")

timestamp = datetime.now(GMT\_8).strftime("%Y-%m-%d %H:%M:%S")

record\_data = [timestamp, question\_for\_answer]

reply\_ws.insert\_rows(row=1, values=record\_data, inherit=True)

print(f"Recorded question: {record\_data}")

###############################################################################

# UI AND FLEX MESSAGE FUNCTIONS

###############################################################################

def create\_category\_and\_common\_features():

"""生成分類選擇的Flex Message"""

print("Generating category and common features message.")

categories = get\_unique\_categories()

category\_bubble = BubbleContainer(

body=BoxComponent(

layout="vertical",

contents=[

TextComponent(

text="請選擇問題分類", weight="bold", size="xl", margin="md"

)

]

+ [

TextComponent(

text=f"{idx + 1}. {category}",

size="md",

color="#4682B4",

wrap=True,

margin="md",

action=MessageAction(label=category, text=f"問題分類: {category}"),

)

for idx, category in enumerate(categories[:10])

],

)

)

return FlexSendMessage(

alt\_text="請選擇問題分類",

contents=CarouselContainer(contents=[category\_bubble]),

)

def create\_flex\_message(title, items, item\_type="category", start\_index=1):

"""生成Flex Message以顯示搜尋結果或分類選項"""

bubbles = []

for i in range(0, len(items), 10):

bubble\_contents = [

TextComponent(text=title, weight="bold", size="xl", margin="md")

]

for idx, item in enumerate(items[i : i + 10], start=start\_index):

label\_text = (

f"{idx}. {item['問題描述'] if item\_type == 'question' else item}"

)

action\_text = (

f"問題: {item['問題描述']}"

if item\_type == "question"

else f"問題分類: {item}"

)

bubble\_contents.append(

TextComponent(

text=label\_text,

size="md",

color="#4682B4",

wrap=True,

margin="md",

action=MessageAction(label=label\_text[:20], text=action\_text),

)

)

bubble\_contents.append(SeparatorComponent(margin="md"))

bubble\_contents.append(

TextComponent(

text="🔙 問題分類",

weight="bold",

color="#228B22",

wrap=True,

action=MessageAction(label="問題分類", text="返回問題分類"),

)

)

bubbles.append(

BubbleContainer(

body=BoxComponent(layout="vertical", contents=bubble\_contents)

)

)

start\_index += 10

print(f"Generated Flex Message with title '{title}' and {len(bubbles)} bubbles.")

return (

FlexSendMessage(

alt\_text="請選擇分類或問題描述",

contents=CarouselContainer(contents=bubbles),

)

if bubbles

else TextSendMessage(text="找不到符合條件的資料。")

)

def build\_flex\_response(answer, conversation\_id):

"""建立包含回饋按鈕的Flex回覆"""

return FlexSendMessage(

alt\_text="回覆與回饋",

contents={

"type": "bubble",

"body": {

"type": "box",

"layout": "vertical",

"contents": [

{"type": "text", "text": answer, "wrap": True},

{

"type": "box",

"layout": "horizontal",

"margin": "md",

"contents": [

{

"type": "button",

"action": {

"type": "postback",

"label": "👍",

"data": f"feedback=thumbs\_up&conv\_id={conversation\_id}",

},

"height": "sm",

"flex": 1,

},

{

"type": "button",

"action": {

"type": "postback",

"label": "👎",

"data": f"feedback=thumbs\_down&conv\_id={conversation\_id}",

},

"height": "sm",

"flex": 1,

},

],

},

],

},

},

)

###############################################################################

# LINE BOT EVENT HANDLERS

###############################################################################

@app.route("/callback", methods=["POST"])

def callback(request):

print(f"Version Code: {VERSION\_CODE}")

signature = request.headers.get("X-Line-Signature")

body = request.get\_data(as\_text=True)

print("Request body:", body)

try:

payload = json.loads(body)

if payload.get("destination") != ALLOWED\_DESTINATION:

print("Invalid destination.")

return "Forbidden", 403

except Exception as e:

print("Payload parsing error:", e)

return "Bad Request", 400

try:

handler.handle(body, signature)

print("Message handled successfully.")

except InvalidSignatureError as e:

print("InvalidSignatureError:", e)

abort(400)

return "OK"

@handler.add(MessageEvent, message=TextMessage)

def handle\_message(event):

user\_input = event.message.text

user\_id = event.source.user\_id

if user\_input.startswith("知識寶典") or user\_input.startswith("返回問題分類"):

reply = create\_category\_and\_common\_features()

print("Displayed category and common features message.")

elif user\_input.startswith("問題分類:"):

category = user\_input.replace("問題分類:", "", 1).strip()

print(f"Processing category request: '{category}'")

questions = get\_questions\_by\_category(category)

if questions:

print(f"Found {len(questions)} questions for category '{category}'")

reply = create\_flex\_message(f"{category} - 問題列表", questions, "question")

else:

print(f"No questions found for category '{category}'")

reply = TextSendMessage(

text=f"找不到「{category}」分類的相關問題。請確認分類名稱是否正確。"

)

elif user\_input.startswith("問題:"):

question = user\_input.replace("問題:", "", 1).strip()

print(f"Looking for solution to question: '{question}'")

solution = find\_solution\_by\_click\_question(question)

if solution:

reply\_contents = [

TextComponent(text="解決方式", weight="bold", size="lg", margin="md"),

TextComponent(

text=solution, size="sm", color="#6A5ACD", wrap=True, margin="md"

),

SeparatorComponent(margin="md"),

TextComponent(

text="🔙 返回問題分類",

weight="bold",

color="#228B22",

wrap=True,

margin="md",

action=MessageAction(label="返回問題分類", text="返回問題分類"),

),

]

reply = FlexSendMessage(

alt\_text="解決方式",

contents=BubbleContainer(

body=BoxComponent(

layout="vertical", contents=reply\_contents, padding\_all="xl"

)

),

)

print(f"Displayed solution for question: {question}")

else:

reply = TextSendMessage(text="找不到該問題的解決方式。")

print(f"No solution found for question: {question}")

elif user\_input == "熱門查詢":

top\_questions = get\_top\_questions()

if top\_questions:

reply = create\_flex\_message("熱門查詢 - Top 5 問題", top\_questions, "question")

else:

reply = TextSendMessage(text="目前沒有熱門排行記錄。")

print("Displayed top 5 questions.")

elif user\_input == "查中油點數":

oil\_points\_message = get\_oil\_points\_column\_a()

reply = TextSendMessage(text=oil\_points\_message)

print("Displayed '中油兌換點數' column A.")

else:

try:

result\_bundle = find\_closest\_question\_and\_llm\_reply(user\_input)

conversation\_id = f"conv\_{user\_id}\_{int(time.time())}"

reply = build\_flex\_response(result\_bundle["answer"], conversation\_id)

print(f"Show LLM answer for question: {user\_input}")

if result\_bundle["top\_matches"]:

top1 = result\_bundle["top\_matches"][0]

db.collection("conversations").add({

"conversation\_id": conversation\_id,

"user\_id": user\_id,

"question": user\_input,

"answer": result\_bundle["answer"],

"matched\_question": top1["question"],

"bm25\_score": top1["bm25\_score"],

"semantic\_score": top1["semantic\_score"],

"combined\_score": top1["combined\_score"],

"model\_version": VERSION\_CODE,

"timestamp": firestore.SERVER\_TIMESTAMP

})

except Exception as e:

print(f"Error in find\_closest\_question\_and\_llm\_reply: {str(e)}")

reply = TextSendMessage(text="機器人暫時無法使用，請聯絡積慧幫忙協助")

try:

line\_bot\_api.reply\_message(event.reply\_token, reply)

print("Reply sent successfully.")

except LineBotApiError as e:

print(f"Failed to send reply: {e}")

# 非同步記錄用戶提問

threading.Thread(target=record\_question, args=(user\_id, user\_input)).start()

@handler.add(PostbackEvent)

def handle\_postback(event):

"""處理用戶回饋"""

data = event.postback.data

params = dict(x.split("=") for x in data.split("&"))

feedback\_type = params.get("feedback")

conversation\_id = params.get("conv\_id")

user\_id = event.source.user\_id

db.collection("feedback").add({

"user\_id": user\_id,

"conversation\_id": conversation\_id,

"feedback\_type": feedback\_type,

"timestamp": firestore.SERVER\_TIMESTAMP

})

line\_bot\_api.reply\_message(

event.reply\_token, TextSendMessage(text="感謝您的回饋 🙏")

)

###############################################################################

# MAIN APPLICATION

###############################################################################

if \_\_name\_\_ == "\_\_main\_\_":

port = int(os.environ.get("PORT", 5000))

print(f"Running on port {port}")

app.run(host="0.0.0.0", port=port)