DAY – 43

29 September 2025

1. **Language detection** mis-identifies Romanized inputs (e.g., “kisan register kive karna”) as English.
2. **Translation prompts** aren’t strict, so Gemini sometimes returns English.

Below is a **fixed, merged implementation** that:

- Detects **Gurmukhi (Punjabi)** and **Devanagari (Hindi)** by script (reliable).
- Falls back to langdetect, plus a few **Romanized Punjabi/Hindi heuristics**.
- Uses **strict translation prompts** (“output only the translation in <language> script, no extra text”).
- Keeps your normalization + misspelling + synonym handling.
- Translates **both ways** (user→English for search, answer→user’s language for reply).
- Adds defensive checks and clearer errors.

---

chatbot_logic.py

```
import os
import re
import unicodedata
import requests
import difflib
import langdetect

# ===== CONFIG =====
HF_API_KEY = os.getenv("HF_API_KEY", "hf_xxxxx")          # <- put your key or use env var
GEMINI_API_KEY = os.getenv("GEMINI_API_KEY", "AIzaSyC-xxxxx")  # <- put your key or use env var
```

```
DATASET_PATH = "dataset.txt"

# ===== LOAD DATASET (English only) =====
faq_data = {}
with open(DATASET_PATH, "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line or "=" not in line:
            continue
        q, a = line.split("=", 1)
        faq_data[q.strip().lower()] = a.strip()

# ===== NORMALIZATION RESOURCES =====
SHORT_FORM_DICT = {
    "u": "you", "ur": "your", "r": "are", "pls": "please", "plz": "please",
    "btw": "by the way", "idk": "i dont know", "thx": "thanks", "ty": "thank you",
    "msg": "message", "info": "information", "asap": "as soon as possible",
    "frm": "from", "abt": "about", "reg": "registration", "signup": "sign up",
    "whts": "what is", "nm": "name", "fr": "for"
}
MISSPELLINGS = {
    "registartion": "registration",
    "regestration": "registration",
    "applcation": "application",
    "adress": "address",
    "teh": "the"
}
SYNONYMS = {
    "sign up": "register",
    "apply": "register",
    "application": "register",
    "enroll": "register",
    "farmer registration": "register as a farmer",
    "registration process": "register process"
```

```
}
STOPWORDS = {
    "the","is","am","are","was","were","a","an","and","or","to","of","in","on","for",
    "with","me","my","i","you","your","please","kindly","hi","hello","hey","what",
    "how","when","where","which","who","whom","do","does","did","can","could","would",
    "should","may","might"
}

# ===== Helpers =====
def _sorted_word_keys(d):
    return sorted(d.keys(), key=lambda k: len(k), reverse=True)

def replace_by_dict(text, mapping):
    for k in _sorted_word_keys(mapping):
        pattern = r'\b' + re.escape(k) + r'\b'
        text = re.sub(pattern, mapping[k], text)
    return text

def cheap_stem(w):
    for suf in ("ing","ed","es","s"):
        if w.endswith(suf) and len(w) > len(suf) + 2:
            return w[:-len(suf)]
    return w

def normalize_text(t):
    t = unicodedata.normalize("NFKC", t).lower()
    t = replace_by_dict(t, SHORT_FORM_DICT)
    t = replace_by_dict(t, MISSPELLINGS)
    t = replace_by_dict(t, SYNONYMS)
    t = re.sub(r'[^a-z0-9\s]', ' ', t)
    t = re.sub(r'\s+', ' ', t).strip()
    return t

def tokenize_and_stem(t):
```

```
    tokens = [cheap_stem(w) for w in t.split() if w and w not in STOPWORDS]
    return set(tokens)

# ===== Robust Language Detection =====
# 1) Script detection (reliable)
#    - Punjabi (Gurmukhi): U+0A00–U+0A7F
#    - Hindi (Devanagari): U+0900–U+097F
# 2) If no script match -> langdetect
# 3) Heuristics for Romanized Punjabi/Hindi
def contains_range(text, start, end):
    return any(start <= ord(ch) <= end for ch in text)

def detect_language_code(user_text: str) -> str:
    t = user_text or ""
    if contains_range(t, 0x0A00, 0x0A7F):
        return "pa"  # Punjabi (Gurmukhi)
    if contains_range(t, 0x0900, 0x097F):
        return "hi"  # Hindi (Devanagari)
    try:
        code = langdetect.detect(t)
    except:
        code = "en"

    # Heuristics for Romanized Punjabi/Hindi
    lower = t.lower()
    roman_pa_markers = ["kive", "kiven", "kidda", "kida", "kithon", "ki", "hanji", "bhenji",
"veerji"]
    roman_hi_markers = ["kaise", "kahan", "kab", "ky", "kya", "hai", "krna", "krdo", "krdena",
"kr diya"]
    if code == "en":  # langdetect often returns en for Romanized
        if any(w in lower for w in roman_pa_markers):
            return "pa"
        if any(w in lower for w in roman_hi_markers):
            return "hi"
```

```
    return code

# ===== Spelling Correction (English only) =====
def correct_spelling(text):
    if not text.strip():
        return text
    url = "https://api-inference.huggingface.co/models/oliverguhr/spelling-correction-english-base"
    headers = {"Authorization": f"Bearer {HF_API_KEY}"}
    payload = {"inputs": text}
    try:
        response = requests.post(url, headers=headers, json=payload, timeout=15)
        if response.status_code == 200:
            return response.json()[0].get('generated_text', text)
    except Exception:
        pass
    return text

# ===== Translation (strict) using Gemini =====
def translate_text(text, target_lang_label: str):
    """
    target_lang_label examples:
      - 'English'
      - 'Punjabi (Gurmukhi script)'
      - 'Hindi (Devanagari script)'
    """
    if not text.strip():
        return text
    url = f"https://generativelanguage.googleapis.com/v1/models/gemini-pro:generateContent?key={GEMINI_API_KEY}"
    headers = {"Content-Type": "application/json"}
    prompt = (
        f"Translate the following text. Output ONLY the translation in {target_lang_label}. "
        f"Do not add any extra words, notes, or quotes.\n\nText:\n{text}"
```

```
    )
    data = {"contents": [{"parts": [{"text": prompt}]}]}
    try:
        res = requests.post(url, headers=headers, json=data, timeout=20)
        if res.status_code == 200:
            j = res.json()
            cand = (j.get("candidates") or [])
            if cand:
                part = cand[0].get("content", {}).get("parts", [{}])[0]
                out = part.get("text", "").strip()
                if out:
                    return out
    except Exception:
        pass
    # Fallback: return original
    return text

# ===== Gemini Smart Fallback (answer builder) =====
def get_gemini_response(prompt):
    if not prompt.strip():
        return "Sorry, I couldn't process your question."
    url = f"https://generativelanguage.googleapis.com/v1/models/gemini-
pro:generateContent?key={GEMINI_API_KEY}"
    headers = {"Content-Type": "application/json"}
    system_prompt = (
        "You are a helpful chatbot for farmers. "
        "If user writes in Punjabi, reply in Punjabi. "
        "If user writes in Hindi, reply in Hindi. "
        "If user says 'in short', give a summary. "
        "If user says 'in detail', give step-by-step. "
        "Otherwise, give clear and simple answer."
    )
    data = {
        "contents": [{
```

```
            "parts": [
                {"text": system_prompt},
                {"text": prompt}
            ]
        }]
    }
    try:
        res = requests.post(url, headers=headers, json=data, timeout=20)
        if res.status_code == 200:
            j = res.json()
            cand = (j.get("candidates") or [])
            if cand:
                part = cand[0].get("content", {}).get("parts", [{}])[0]
                out = part.get("text", "").strip()
                if out:
                    return out
    except Exception:
        pass
    return "Sorry, I couldn't process your question."

# ===== Dataset search =====
def search_dataset(user_input):
    questions = list(faq_data.keys())
    matches = difflib.get_close_matches(user_input.lower(), questions, n=1, cutoff=0.6)
    if matches:
        return faq_data[matches[0]]
    return None

# ===== Main response =====
def get_response(user_input: str) -> str:
    try:
        # 1) Detect language (script first, then langdetect + romanized heuristics)
        lang_code = detect_language_code(user_input)  # 'en', 'pa', 'hi', etc.
```

```
    # 2) If not English, translate to English for dataset search (strict)
    if lang_code != "en":
        target = "English"
        user_for_search = translate_text(user_input, target_lang_label=target)
    else:
        user_for_search = user_input

    # 3) Normalize + spelling correction (English-only normalization)
    user_for_search = normalize_text(correct_spelling(user_for_search))

    # 4) Dataset search
    dataset_answer = search_dataset(user_for_search)

    # 5) Fallback → Gemini
    if not dataset_answer:
        dataset_answer = get_gemini_response(user_for_search)

    # 6) Translate answer back to original language if needed
    if lang_code == "pa":
        return translate_text(dataset_answer, target_lang_label="Punjabi (Gurmukhi script)")
    elif lang_code == "hi":
        return translate_text(dataset_answer, target_lang_label="Hindi (Devanagari script)")
    else:
        return dataset_answer

  except Exception as e:
    # Final safety net so the API never crashes your Flask app
    return f"Sorry, something went wrong: {e}"
```

---

app.py

```
from flask import Flask, render_template, request, jsonify
from chatbot_logic import get_response
```

```
app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")  # make sure templates/index.html exists

@app.route("/get", methods=["POST"])
def chatbot_reply():
    try:
        data = request.get_json(force=True, silent=True) or {}
        user_message = data.get("msg", "").strip()
        if not user_message:
            return jsonify({"response": "Please type a message."}), 200
        reply = get_response(user_message)
        return jsonify({"response": reply}), 200
    except Exception as e:
        # Never crash the server; return a readable error to the UI
        return jsonify({"response": f"Server error: {e}"}), 200

if __name__ == "__main__":
    # For local testing only. In production, use a WSGI server (gunicorn/uwsgi).
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Why this works now

- **Punjabi/Hindi detection**
  - If the user types in **Gurmukhi** or **Devanagari**, we **force** pa/hi.
  - If they type **Romanized**, heuristics catch common words (e.g., *kive, kaise, hai*) and force pa/hi.
  - Otherwise, fall back to langdetect.
- **Strict translations**

  - Prompts instruct Gemini to **output only the translation** in the **correct script**; this stops English fallbacks.
- **Resilience**
  - Defensive JSON parsing and exception handling so your Flask route doesn't crash.
  - If any network/API issue occurs, user still gets a readable message.

Quick checklist (common gotchas)

- Install deps:
- pip install flask requests langdetect
- Ensure your file layout:
- project/
- app.py
- chatbot_logic.py
- dataset.txt
- templates/
- index.html
- static/
- style.css
- script.js
- Put **valid** API keys in env or in the file:
- export HF_API_KEY=your_hf_key
- export GEMINI_API_KEY=your_gemini_key
- dataset.txt must be **English-only** with lines like:
- what is your name = I am your assistant chatbot.
- tell me about new registration = To use the application, first click the sign-up button...

  (spelling of "registration" matters, but we added misspelling normalization.)