DAY – 69

10 November 2025

This into your file (replace the current ML / matching functions: classify_intent, ml_best_match, rule_based_match and the matching section inside get_response). It's a **drop-in** — nothing else needs to change.

```
# ---------- Ensemble matching (drop-in replacement) ----------
from difflib import SequenceMatcher
import numpy as np

def fuzzy_similarity(a, b):
    """Return 0..1 fuzzy similarity using SequenceMatcher (handles paraphrase-ish cases)."""
    if not a or not b:
        return 0.0
    return SequenceMatcher(None, a, b).ratio()

def find_best_answer(user_norm, user_tokens, debug=False):
    """
    Ensemble matcher: combines TF-IDF similarity, classifier prob, rule-based score, and fuzzy ratio.
    Returns (best_answer_or_None, combined_score_float).
    """
    best_ans = None
    best_score = 0.0

    # safety: need index/vectorizer/tfidf_matrix/answers
    has_tfidf = vectorizer is not None and tfidf_matrix is not None and len(answers) > 0
    has_classifier = classifier_pipeline is not None
    has_index = bool(index.get("english"))

    # Precompute TF-IDF sims (if available)
    tfidf_sims = None
    if has_tfidf:
```

```
    try:
        qvec = vectorizer.transform([user_norm])
        tfidf_sims = cosine_similarity(qvec, tfidf_matrix)[0]
    except Exception as e:
        if DEBUG:
            print("TF-IDF sim error:", e)
        tfidf_sims = None

# Loop through indexed Qs (answers list) and compute component scores
for idx, ans in enumerate(answers):
    comp_scores = []

    # 1) TF-IDF similarity component (0..1)
    tfidf_score = 0.0
    if tfidf_sims is not None:
        try:
            tfidf_score = float(tfidf_sims[idx])
        except Exception:
            tfidf_score = 0.0
    comp_scores.append(("tfidf", tfidf_score))

    # 2) Classifier probability that this index is predicted (0..1)
    clf_score = 0.0
    if has_classifier:
        try:
            probs = classifier_pipeline.predict_proba([user_norm])[0]
            # classifier was trained with labels 0..N-1 in same order as answers
            if idx < len(probs):
                clf_score = float(probs[idx])
            else:
                # improbable — fallback to max prob as weak signal
                clf_score = float(max(probs))
        except Exception as e:
            if DEBUG:
```

```
            print("Classifier prob error:", e)
        clf_score = 0.0
    comp_scores.append(("clf", clf_score))

    # 3) Rule-based overlap score (0..1) using token sets (existing rule-based method but normalized)
    rb_score = 0.0
    if has_index:
        try:
            e = index["english"][idx]
            tokens_e = e.get("tokens", set())
            if not tokens_e:
                rb_score = 0.0
            else:
                union = tokens_e | user_tokens
                if union:
                    rb_score = len(tokens_e & user_tokens) / len(union)
        except Exception as e:
            if DEBUG:
                print("RB score error:", e)
            rb_score = 0.0
    comp_scores.append(("rb", rb_score))

    # 4) Fuzzy textual similarity between normalized questions (catches paraphrase)
    fuzzy_score = 0.0
    try:
        nq = index["english"][idx]["nq"] if has_index else normalize_text(questions[idx])
        fuzzy_score = fuzzy_similarity(user_norm, nq)
    except Exception:
        fuzzy_score = 0.0
    comp_scores.append(("fuzzy", fuzzy_score))

    # Weighted combination: tune these weights as needed.
    # TF-IDF is strongest, then classifier, then rule, then fuzzy as supplement.
```

```
        weights = {"tfidf": 0.40, "clf": 0.30, "rb": 0.18, "fuzzy": 0.12}

        combined = 0.0
        for name, val in comp_scores:
            combined += weights.get(name, 0.0) * max(0.0, min(1.0, float(val)))

        if debug:
            print(f"[candidate {idx}] tfidf={tfidf_score:.3f} clf={clf_score:.3f} rb={rb_score:.3f}
fuzzy={fuzzy_score:.3f} => combined={combined:.3f}")

        if combined > best_score:
            best_score = combined
            best_ans = ans

    # Normalize best_score to 0..1 range (already in 0..1)
    return best_ans, float(best_score)



# ---------- How to use inside get_response ----------
# Replace the current TF-IDF / rule-based / classifier selection block with this:
#
#   # previous: best_answer, best_score = None, 0.0
#   best_answer, best_score = find_best_answer(user_norm, user_tokens, debug=DEBUG)
#
#   # If best_score is low, try RAG as fallback (keeps your existing RAG logic)
#   if not best_answer or best_score < 0.30:
#       try:
#           rag_input = f"{query}\nContext: {' '.join([u for u, _ in history[-3:]])}"
#           rag_ans = get_rag_response(rag_input, chat_history=history)
#           if rag_ans and not rag_ans.lower().startswith(("sorry", "couldn't", "i cannot")):
#               best_answer = rag_ans
#               best_score = max(best_score, 0.85)
#       except Exception as e:
#           if DEBUG:
```

```
#             print("RAG error:", e)
#
#   # if still no good answer, fallback to your default message
#   if not best_answer:
#       best_answer = "Sorry, I don’t have an answer for that yet. Could you specify the topic again?"
#
# That small replacement removes brittle single-signal thresholds and gives you robust matching.
```

Why this should fix your "keyword-only" problem

- TF-IDF alone misses paraphrases. Fuzzy (SequenceMatcher) catches paraphrase / rewordings and gives them a nonzero score.
- Classifier probability contributes when it is confident, but doesn’t block matches when low.
- Rule-based token overlap still helps exact-phrase matches.
- The ensemble uses **weighted combination**, so any single signal being low doesn’t throw the whole decision away.
- The RAG fallback is unchanged but now triggers only if ensemble confidence is low (< 0.30) — you can lower/raise that threshold as needed.

Quick tuning suggestions

- If you still get false positives, raise the final RAG threshold (e.g., require `best_score >= 0.40` to accept an indexed answer).
- If you miss paraphrases, increase `weights["fuzzy"]` to ~0.25 and reduce TF-IDF weight a bit.
- Turn `DEBUG = True` and try several example queries to watch component scores printed, then tune weights.