DAY – 61 to 63

12 to 14 November 2025

Possible Causes

1. **Text wrap**: UI framework (Flutter/React/HTML) automatic wrap kar rahi aa, jis naal alignment hil rahi hai.
2. **Line breaks**: Answer string vich \n ya unwanted spaces/joining characters han.
3. **Font mismatch**: Grey box te green box different font-size / padding use kar rahe han.
4. **Container styling issue**: Justification left hai, center nahi.

Solutions

Agar **HTML/CSS/React** use kar rahe ho:

```
.answer-box {
  text-align: justify;  /* ya left, jo tusi chahunde ho */
  white-space: pre-line;  /* \n nu respect karega */
  line-height: 1.5;  /* spacing sudharan lai */
  padding: 10px;
}
```

Agar **Flutter** use kar rahe ho:

```
Text(
  answer,
  textAlign: TextAlign.justify,  // ya TextAlign.left
  style: TextStyle(
    height: 1.5, // line spacing
  ),
)
```

Agar **Python/Streamlit** use kar rahe ho:

```
st.markdown(
    f"<div style='text-align: justify; line-height:1.5'>{answer}</div>",
    unsafe_allow_html=True
)
```

```
/* Messages */

.chat-message {

  padding: 10px 14px;

  border-radius: 18px;

  max-width: 77%;

  word-wrap: break-word;

  white-space: pre-line;   /* line breaks (\n) nu respect karega */

  line-height: 1.5;

  margin-bottom: 6px;

  font-size: 14px;

}

.bot {

  background-color: #e0e0e0;

  align-self: flex-start;

  text-align: justify;     /* text justify ho ke saaf align hovega */

  color: #000;

}
```

```
.user {
  background-color: #4CAF50;
  color: white;
  align-self: flex-end;
  text-align: left;
}
```

Ki badaleya?

- white-space: pre-line; → line breaks from backend (e.g., \n) ab UI vich dikhange.
- .bot { text-align: justify; } → grey box vich text left-right dono taraf equal spacing naal aaega.
- line-height: 1.5; → thoda clean spacing.

Solution

Main tuhade load_and_train() function nu modify karda, taki:

1. **dataset.txt read kare**
    - Format: Question ::: Answer har line vich.
    - Punjabi te English dono support.
2. **faq_data["english"]** + **faq_data["punjabi"]** v update kare.
3. PDF + Dataset dono Q/A merge karke training ho jave.

Fixed Code Snippet (replace your load_and_train)

```
def load_and_train():
    global faq_data, questions, answers, classifier_pipeline, vectorizer, tfidf_matrix, index
    global last_modified_pdf, last_modified_dataset

    # reset data holders
    faq_data = {"english": {}, "punjabi": {}}
    questions, answers = [], []

    # ===== Load from PDF =====
    parsed_from_pdf = {}
    if os.path.exists(PDF_PATH):
        pdf_text = robust_extract_pdf_text(PDF_PATH)
        parsed_from_pdf = parse_pdf_qa_strict(pdf_text)

        if DEBUG:
            print(f"[debug] PDF text length: {len(pdf_text)} chars")
            print(f"[debug] Parsed {len(parsed_from_pdf)} Q/A from PDF")

        if parsed_from_pdf:
            faq_data["english"].update(parsed_from_pdf)

    # ===== Load from dataset.txt =====
    if os.path.exists(DATASET_PATH):
        with open(DATASET_PATH, "r", encoding="utf-8") as f:
            for line in f:
                if " ::: " in line:
                    q, a = line.split(" ::: ", 1)
                    q, a = q.strip(), a.strip()
                    if not q or not a:
                        continue
                    # Detect if Punjabi or English
                    if re.search(r'[\u0A00-\u0A7F]', q+a):  # Punjabi Unicode range
                faq_data["punjabi"][q.lower()] = a
            else:
                faq_data["english"][q.lower()] = a

        if DEBUG:
            print(f"[debug] Loaded {len(faq_data['english'])} EN + {len(faq_data['punjabi'])} PA
Q/A from dataset.txt")

        last_modified_dataset = os.path.getmtime(DATASET_PATH)

    # ===== Build ML pipeline only if we got Q/As =====
    questions = list(faq_data["english"].keys())
    answers = list(faq_data["english"].values())

    if questions:
        norm_questions = [normalize_text(q) for q in questions]

        # intent classifier
        classifier_pipeline = make_pipeline(
            TfidfVectorizer(),
            LogisticRegression(max_iter=2000)
        )
        classifier_pipeline.fit(norm_questions, list(range(len(questions))))

        # tf-idf similarity index
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(norm_questions)

        # token index for rule-based matching
        index = {"english": []}
        for q, a in faq_data["english"].items():
            nq = normalize_text(q)
            toks = tokenize_and_stem(nq)
            index["english"].append({"q": q, "nq": nq, "tokens": toks, "a": a})
    last_modified_pdf = os.path.getmtime(PDF_PATH) if os.path.exists(PDF_PATH) else
None

    print(f" Loaded {len(questions)} EN Q/A + {len(faq_data['punjabi'])} PA Q/A")
```

Example dataset.txt format

```
What is e-Sinchai? ::: e-Sinchai is an irrigation management system...
Who can apply for PISMS? ::: Farmers in Punjab can apply online...
e-Sinchai kiven kam karda hai? ::: Eh system remote sensing data nal irrigation manage karda
hai.
```

- merge_wrapped_lines → broken lines nu properly merge karan layi.
- is_heading → all-caps ya short headings nu detect karan layi.
- SKIP_PATTERNS → tables/figures/annexures jive unwanted cheezan skip karan layi.
- OCR fallback nu per-page optimize kita.

Optimized Version

```
# ===== Extra skip patterns =====
SKIP_PATTERNS = [
    r"^\s*Table\s*\d+",
    r"^\s*Figure\s*\d+",
    r"^\s*Fig\.\s*\d+",
    r"^\s*Annexure",
    r"^\s*Appendix",
]
```

```
# ===== Merge wrapped lines =====
def merge_wrapped_lines(lines):
    merged, buffer = [], ""
    for l in lines:
        if not l.strip():
            if buffer:
                merged.append(buffer.strip())
                buffer = ""
            continue
        # continuation if prev not ended with full stop/punctuation
        if buffer and not buffer.endswith((".", "?", "!", ":")) and l[0].islower():
            buffer += " " + l
        else:
            if buffer:
                merged.append(buffer.strip())
            buffer = l
    if buffer:
        merged.append(buffer.strip())
    return merged

# ===== Heading detector =====
def is_heading(line):
    words = line.strip().split()
    return (line.isupper() and 1 <= len(words) <= 7) or re.match(r'^\d+(\.|\))', line)

# ===== Improved robust_extract_pdf_text =====
def robust_extract_pdf_text(pdf_path):
    text = ""
    if not os.path.exists(pdf_path):
        return ""
    try:
        with open(pdf_path, "rb") as f:
            reader = PyPDF2.PdfReader(f)
            for i, page in enumerate(reader.pages):
```

```
        ptext = page.extract_text()
        if not ptext or len(ptext.strip()) < 20:
            # OCR fallback only for this page
            try:
                img = convert_from_path(pdf_path, dpi=200, first_page=i+1,
last_page=i+1)[0]
                ptext = pytesseract.image_to_string(img)
                if DEBUG:
                    print(f"[OCR fallback] used on page {i+1}")
            except Exception as e:
                if DEBUG:
                    print("OCR error on page", i+1, e)
                continue

        lines = [l.rstrip() for l in ptext.splitlines()]
        clean_lines = []
        for l in lines:
            if not l.strip():
                continue

            # Skip headers/footers
            if any(re.match(pat, l, flags=re.I) for pat in HEADER_FOOTER_PATTERNS +
SKIP_PATTERNS):
                continue

            # TOC style dotted leader removal
            m = re.search(r'^(.*?)\s*\.{3,}\s*(\d{1,4})\s*$', l)
            if m:
                l = m.group(1).rstrip()
                if not l:
                    continue

            # Remove junk (only dots/underscores etc.)
            if re.match(r'^[\s\.\-——\_]{3,}$', l):
                continue

            # Remove trailing page no
            l = re.sub(r'\s{2,}\d{1,4}\s*$', '', l)

            # Skip high punctuation ratio
            punct_count = sum(1 for ch in l if not ch.isalnum() and not ch.isspace())
            if len(l) > 0 and (punct_count / len(l)) > 0.45:
                continue

            clean_lines.append(l)

        # Merge wrapped lines before saving
        clean_lines = merge_wrapped_lines(clean_lines)

        if clean_lines:
            text += "\n".join(clean_lines) + "\n\n"
    except Exception as e:
        if DEBUG:
            print("PyPDF2 extraction error:", e)

    return text

# ===== Improved parse_pdf_qa_strict =====
def parse_pdf_qa_strict(text):
    faqs = {}
    if not text or not text.strip():
        return faqs

    raw_lines = [l.rstrip() for l in text.splitlines()]
    i, n = 0, len(raw_lines)

    while i < n:
        line = raw_lines[i].strip()
        next_line = raw_lines[i+1] if i+1 < n else ""
        sc = score_line_question(line, next_line)

        if sc >= 1.2 or line.lower().startswith(("steps to", "procedure", "how to", "guide")) or
is_heading(line):
            q = line.strip(":").strip()
            a_lines, j = [], i+1

            while j < n:
                ln = raw_lines[j].strip()
                if not ln:
                    j += 1
                    continue
                if score_line_question(ln, raw_lines[j+1] if j+1 < n else "") >= 1.2 or
is_heading(ln):
                    break
                if any(re.match(pat, ln, flags=re.I) for pat in HEADER_FOOTER_PATTERNS +
SKIP_PATTERNS):
                    j += 1
                    continue

                # Keep bullets/steps
                if ln.startswith(("➢", "•", "-", "▪", "Step", "→")):
                    a_lines.append(ln)
                else:
                    a_lines.append(ln)

                j += 1

            ans = "\n".join(a_lines).strip()
            if ans:
                faqs[q.lower()] = ans
            i = j
        else:
            i += 1

    # Debug dump
    if DEBUG:
        with open("parsed_debug.txt", "w", encoding="utf-8") as dbg:
            for q, a in faqs.items():
                dbg.write("Q: " + q + "\nA: " + a + "\n\n")

    return faqs
```

Key Fixes:

- **Line merging** → no more half-sentences.
- **Heading detection** → more reliable Q/A boundaries.
- **Selective OCR** → only on problematic pages (fast + reliable).
- **Skip patterns** → tables/figures/annexures ignore ho jaange.
- **Debug file** → easily check karo parse hoeya data.

- answer collection nu **stop conditions** naal tighten karde ne (max chars, new-section patterns, ALL-CAPS headings, numbered headings, big punctuation runs).
- TOC/dotted-leader handling better kar ditta (tusi pehlan wekkheya si — keep that).
- bullets/steps nu preserve karde hoye long paragraphs nu **chunk** karna (short summary or limit).
- debug prints optional (DEBUG flag use karo).

Copy-paste karo eh updated parse_pdf_qa_strict + small format_answer tweak — replace existing function(s) with these.

```
# ===== Revised parse_pdf_qa_strict (with answer-length limits + improved break rules) =====
def parse_pdf_qa_strict(text, max_answer_chars=1400, max_answer_lines=60):
    """
    Improved detection: stop collecting answer when:
     - we hit another likely question (score >= 1.2),
     - or we hit a strong section heading (ALL CAPS, numbered heading, 'References' etc.),
     - or answer exceeds max_answer_chars or max_answer_lines.
    Returns dict mapping lowercased question -> answer (raw).
    """
    faqs = {}
    if not text or not text.strip():
        return faqs

    raw_lines = [l.rstrip() for l in text.splitlines()]
    i, n = 0, len(raw_lines)

    # helper patterns
    ALL_CAPS_RE = re.compile(r'^[\s\W]*[A-Z0-9 \-\/:&]{4,}$')  # lines that look like section headings
    NUM_HEADING_RE = re.compile(r'^\s*(?:\d+(\.\d+)*\s*[-\)]?\s+)')  # 1. or 1.1 or 2)
    SECTION_BREAK_RE = re.compile(r'^(References|Bibliography|Annex|Appendix|Index|Acknowledg)', flags=re.I)
    FIG_TABLE_RE = re.compile(r'\b(Figure|Fig|Diagram|Table)\b', flags=re.I)
```

```
while i < n:
    line = raw_lines[i].strip()
    next_line = raw_lines[i+1] if i+1 < n else ""
    sc = score_line_question(line, next_line)

    # treat short headings like "Steps", "Procedure" as questions too
    if sc >= 1.2 or line.lower().startswith(("steps to", "procedure", "how to", "guide")):
        q = line.strip(":").strip()
        # normalize trivial Q text
        if not q:
            i += 1
            continue

        a_lines, j = [], i+1
        chars_collected = 0
        lines_collected = 0

        while j < n:
            ln = raw_lines[j].strip()
            if not ln:
                j += 1
                continue

            # immediate break conditions
            if score_line_question(ln, raw_lines[j+1] if j+1 < n else "") >= 1.2:
                if DEBUG:
                    print("[parse_pdf_qa_strict] breaking: next likely question:", ln)
                break
            if SECTION_BREAK_RE.match(ln):
                if DEBUG:
                    print("[parse_pdf_qa_strict] breaking: section end:", ln)
                break
            # skip figure/table captions (they rarely belong to textual answer)
            if FIG_TABLE_RE.search(ln):
                if DEBUG:
                    print("[parse_pdf_qa_strict] skipping figure/table line")
                j += 1
                continue

            # skip header/footer matches
            skip = False
            for pat in HEADER_FOOTER_PATTERNS:
                if re.match(pat, ln, flags=re.I):
                    skip = True
                    break
            if skip:
                j += 1
                continue

            # strong heading / new subsection -> probably end of answer
            if NUM_HEADING_RE.match(ln) or (ALL_CAPS_RE.match(ln) and
    len(ln.split()) <= 8):
                if DEBUG:
                    print("[parse_pdf_qa_strict] breaking: detected new subsection or ALL-CAPS
    heading:", ln)
                break

            # punctuation-only lines skip
            if re.match(r'^[\s\.\-——\_]{3,}$', ln):
                j += 1
                continue

            # collect bullets or normal lines
            if ln.startswith(("➢", "•", "-", "▪", "Step", "→")) or len(ln.split()) >= 1:
                a_lines.append(ln)
```

```
                chars_collected += len(ln)
                lines_collected += 1

            # stop if answer becomes too big (prevents swallowing whole doc)
            if chars_collected > max_answer_chars or lines_collected > max_answer_lines:
                if DEBUG:
                    print(f"[parse_pdf_qa_strict] stopping collection (size limit)
chars={chars_collected} lines={lines_collected}")
                break

            j += 1

        ans = "\n".join(a_lines).strip()
        if ans:
            faqs[q.lower()] = ans
        i = j
    else:
        i += 1

return faqs


# ===== Small tweak to format_answer: if paragraph > X chars, break into smaller paras
(preserve bullets) =====
def format_answer(ans, max_para_chars=800):
    if not ans:
        return ans

    paras = re.split(r'\n\s*\n', ans.strip())
    formatted = []

    for para in paras:
        lines = [l.strip() for l in para.splitlines() if l.strip()]
        if not lines:
```

```
        continue

    # keep bullets as before
    if all(l.startswith(("•", "-", "➢", "▪", "Step", "→")) for l in lines):
        formatted.extend(lines)
        formatted.append("")
        continue

    # if a single paragraph is huge, try to split on sentence boundaries (~". ")
    merged = " ".join(lines)
    if len(merged) > max_para_chars:
        # naive sentence split
        sents = re.split(r'(?<=[\.\?\!])\s+', merged)
        cur = []
        cur_len = 0
        for s in sents:
            if cur_len + len(s) > max_para_chars and cur:
                formatted.append(" ".join(cur))
                formatted.append("")
                cur = [s]
                cur_len = len(s)
            else:
                cur.append(s)
                cur_len += len(s)
        if cur:
            formatted.append(" ".join(cur))
        formatted.append("")
        continue

    # normal para
    formatted.append(merged)
    formatted.append("")
```

return "\n".join(formatted).strip()

Hor improvements / suggestions (quick list)

1. **Tune thresholds**: score_line_question cutoff (1.2) is okay, par kise docs ch change karke 1.0 try karo. DEBUG true rakh ke dekho.
2. **Max sizes**: max_answer_chars=1400 te max_answer_lines=60 default rakhe ne — tusi eh ghata ke 600/20 kar sakde ho for shorter answers.
3. **Post-process long answers**: Je parsed answer fir vi long aa, tusi LLM summary karwa ke short version store kar sakde (par LLM call costly). Example: if len(ans)>1000 → call get_gemini_response("Summarize: "+ans, detailed=False).
4. **Indexing granularity**: instead of storing huge answers in FAQ, consider splitting long answers into smaller chunks and index each chunk separately (helps ML + cosine similarity).
5. **TOC lines**: tusi already strip dotted leaders — good. Add removal for lines that look like "Chapter 3 — IRRIGATION" (NUM_HEADING_RE + ALL_CAPS).
6. **OCR noise**: OCR fallback sometimes outputs many stray lines; consider running a quick filter to remove lines with low alpha ratio or too short (<3 chars) before parsing.

How to test quickly

- enable DEBUG=True, run load_and_train() and watch debug prints to see where breaks happen.
- print len(parsed_from_pdf) and sample a few keys/answers:
- parsed_from_pdf = parse_pdf_qa_strict(pdf_text)
- for q,a in list(parsed_from_pdf.items())[:10]:
-     print("Q:", q)
-     print("A preview:", a[:400].replace("\n"," | "))
- Try lowering max_answer_chars to 600 and re-run to see differences.