SupportBot Multimodal Fix Proposal

Pseudoalgorithms for Full Flow Improvement

Technical Report

February 8, 2026

Abstract

This document presents the proposed changes to SupportBot's architecture to support multimodal (text + images) processing. The current system loses visual information by converting images to text descriptions early in the pipeline. The proposed fix sends raw images to the multimodal LLM (Gemini) at decision and response stages while maintaining text-only embeddings for retrieval (Option B approach). Additionally, we address the 43% garbage case rate in the knowledge base.

Contents

1 Executive Summary

1.1 Current Problems Identified

1. **43% garbage cases**: Nearly half of extracted cases have empty `solution_summary`
2. **Images reduced to text**: Visual context (screenshots, error dialogs, hardware photos) is lost at ingestion
3. **Gate sees text only**: `decide_consider()` cannot understand "look at this error" messages
4. **Responder sees text only**: `decide_and_respond()` cannot reason about what user is showing
5. **Eval shows 8.7% pass rate**: On "answer" label messages where bot should have helped

1.2 Proposed Changes (Priority Order)

Priority	Change	Effort	Impact
P0	Reject cases without solutions	Low	High
P1	Pass images to `decide_and_respond()`	Medium	High
P2	Pass images to `decide_consider()`	Medium	Medium
P3	Store image paths in `raw_messages`	Low	Enables P1/P2
P4	Include images in KB case evidence	Medium	Medium

Table 1: Proposed changes by priority

2 Current Algorithm (As-Is)

For reference, here is the current flow with problems highlighted.

2.1 Algorithm 1: Message Ingestion (Current)

Algorithm 1 Current Message Ingestion — Loses image data

procedure IngestMessage($msg\_id, group\_id, sender, ts, text, image\_paths$)
  $content\_text \leftarrow text$
  for $path$ **in** $image\_paths$ **do**
    $img\_bytes \leftarrow$ ReadFile($path$)
    $extraction \leftarrow$ LLM.ImageToText($img\_bytes, text$) ▷ Lossy!
    ▷ Returns {observations: [...], extracted_text: "..."}
    $content\_text \leftarrow content\_text +$ "[image]" + JSON($extraction$)
  // Original images discarded after this point!
  InsertRawMessage($msg\_id, group\_id, ts, hash(sender), content\_text$)
  EnqueueJob($BUFFER\_UPDATE, \{group\_id, msg\_id\}$)
  EnqueueJob($MAYBE\_RESPOND, \{group\_id, msg\_id\}$)

2.2 Algorithm 2: Case Extraction (Current)

Algorithm 2 Current Case Extraction — Creates garbage cases

```
1:  procedure HandleBufferUpdate(group_id, msg_id)
2:      msg ← GetRawMessage(msg_id)
3:      line ← FormatBufferLine(msg)
4:      buffer ← GetBuffer(group_id)
5:      buffer_new ← buffer + line
6:
7:      extract ← LLM.ExtractCase(buffer_new)
8:      if ¬extract.found then
9:          SetBuffer(group_id, buffer_new)
10:         return
11:
12:     case ← LLM.MakeCase(extract.case_block)
13:     if ¬case.keep then
14:         SetBuffer(group_id, extract.buffer_new)
15:         return
16:
17:     // BUG: No validation that solution_summary is non-empty!
18:     case_id ← NewUUID()
19:     InsertCase(case_id, group_id, case.*)
20:
21:     doc_text ← case.problem_title + case.problem_summary + case.solution_summary
22:     embedding ← LLM.Embed(doc_text)                          ▷ Text-only
23:     Chroma.Upsert(case_id, doc_text, embedding, {group_id})
24:     SetBuffer(group_id, extract.buffer_new)
```

2.3 Algorithm 3: Response Pipeline (Current)

Algorithm 3 Current Response Pipeline — Text-only, no images

```
1:  procedure HandleMaybeRespond(group_id, msg_id)
2:      msg ← GetRawMessage(msg_id)
3:      context ← GetLastNMessages(group_id, n)                  ▷ Text only
4:
5:      if ¬MentionsBot(msg.content_text) then
6:          decision ← LLM.DecideConsider(msg.content_text, context)   ▷ No images!
7:          if ¬decision.consider then
8:              return                                           ▷ Silent exit
9:
10:     query_emb ← LLM.Embed(msg.content_text)                  ▷ Text-only
11:     cases ← Chroma.Query(query_emb, k, group_id)
12:
13:     resp ← LLM.DecideAndRespond(msg.content_text, context, cases)   ▷ No images!
14:     if resp.respond then
15:         Signal.SendGroupText(group_id, resp.text)
```

3 Proposed Algorithm (To-Be)

3.1 Algorithm 1': Message Ingestion (Proposed)

Key change: **Store image paths** in database for later retrieval.

Algorithm 4 Proposed Message Ingestion — Preserves image references

procedure IngestMessage($msg\_id, group\_id, sender, ts, text, image\_paths$)
  $content\_text \leftarrow text$
  $stored\_image\_paths \leftarrow []$ ▷ NEW: Track stored images

  for $path$ **in** $image\_paths$ **do**
    $img\_bytes \leftarrow$ ReadFile($path$)
    ▷ Still extract text for embedding/search purposes
    $extraction \leftarrow$ LLM.ImageToText($img\_bytes, text$)
    $content\_text \leftarrow content\_text +$ "[image]" + JSON($extraction$)

    // NEW: Store canonical path for later retrieval
    $canonical\_path \leftarrow$ CanonicalImagePath($path$)
    $stored\_image\_paths \leftarrow stored\_image\_paths + [canonical\_path]$

  // NEW: Store image_paths_json in raw_messages table
  InsertRawMessage($msg\_id, group\_id, ts, hash(sender), content\_text,$
    JSON($stored\_image\_paths$)) ▷ NEW field

  EnqueueJob($BUFFER\_UPDATE, \{group\_id, msg\_id\}$)
  EnqueueJob($MAYBE\_RESPOND, \{group\_id, msg\_id\}$)

3.2 Algorithm 2': Case Extraction (Proposed)

Key changes:

- **Reject cases without solutions**
- **Store image paths in case evidence**

Algorithm 5 Proposed Case Extraction — Validates quality, preserves images

```
1:  procedure HandleBufferUpdate(group_id, msg_id)
2:      msg ← GetRawMessage(msg_id)
3:      line ← FormatBufferLine(msg)
4:      buffer ← GetBuffer(group_id)
5:      buffer_new ← buffer + line
6:
7:      extract ← LLM.ExtractCase(buffer_new)
8:      if ¬extract.found then
9:          SetBuffer(group_id, buffer_new)
10:         return
11:
12:     case ← LLM.MakeCase(extract.case_block)
13:     if ¬case.keep then
14:         SetBuffer(group_id, extract.buffer_new)
15:         return
16:
17:     // NEW: Validate that solved cases have actual solutions
18:     if case.status = "solved" ∧ IsEmpty(case.solution_summary) then
19:         Log.Warning("Rejecting solved case without solution")
20:         SetBuffer(group_id, extract.buffer_new)
21:         return                                          ▷ Reject garbage
22:
23:     case_id ← NewUUID()
24:
25:     // NEW: Collect image paths from evidence messages
26:     evidence_image_paths ← []
27:     for evidence_msg_id in case.evidence_ids do
28:         evidence_msg ← GetRawMessage(evidence_msg_id)
29:         if evidence_msg ≠ null then
30:             evidence_image_paths ← evidence_image_paths + evidence_msg.image_paths
31:
32:     InsertCase(case_id, group_id, case.∗, evidence_image_paths)
33:
34:     doc_text ← case.problem_title + case.problem_summary + case.solution_summary
35:     embedding ← LLM.Embed(doc_text)
36:     Chroma.Upsert(case_id, doc_text, embedding,
37:                   {group_id, evidence_image_paths})     ▷ Include paths in metadata
38:     SetBuffer(group_id, extract.buffer_new)
```

3.3 Algorithm 3': Response Pipeline (Proposed)

Key changes:

- **Load and pass images to gate**
- **Load and pass images to responder**
- **Include KB case images in responder context**

Algorithm 6 Proposed Response Pipeline — Multimodal gate and responder

```
procedure HandleMaybeRespond(group_id, msg_id)
    msg ← GetRawMessage(msg_id)
    context ← GetLastNMessages(group_id, n)

    // NEW: Load images from current message
    msg_images ← []
    for path in msg.image_paths do
        if FileExists(path) then
            msg_images ← msg_images + [ReadFile(path)]

    if ¬MentionsBot(msg.content_text) then
        // NEW: Pass images to gate (multimodal Gemini)
        decision ← LLM.DecideConsider(msg.content_text, context, msg_images)
        if ¬decision.consider then
            return

    query_emb ← LLM.Embed(msg.content_text)                    ▷ Still text-only
    cases ← Chroma.Query(query_emb, k, group_id)

    // NEW: Load images from retrieved KB cases (optional, for evidence)
    kb_images ← []
    for case in cases do
        for path in case.metadata.evidence_image_paths do
            if FileExists(path) ∧ |kb_images| < MAX_KB_IMAGES then
                kb_images ← kb_images + [ReadFile(path)]

    // NEW: Pass all images to responder (multimodal Gemini)
    all_images ← msg_images + kb_images
    resp ← LLM.DecideAndRespond(msg.content_text, context, cases, all_images)
    if resp.respond then
        Signal.SendGroupText(group_id, resp.text)
```

4 LLM Client Changes

4.1 Algorithm 4': Multimodal Decision Gate

Algorithm 7 Proposed `decide_consider()` — Accepts images

```
function LLM.DecideConsider(message, context, images)
    system ← P_DECISION_SYSTEM                    ▷ Same prompt, Gemini handles images
    user_content ← []

    // Build multimodal content array
    user_content ← user_content + [{"type" : "text", "text" : "MESSAGE:" + message}]

    for img_bytes in images do
        b64 ← Base64Encode(img_bytes)
        user_content ← user_content + [{
              "type" : "image_url",
              "image_url" : {"url" : "" + b64}
        }]

    user_content ← user_content + [{"type" : "text", "text" : "CONTEXT:" + context}]

    response ← Gemini.ChatCompletion(system, user_content)
    return ParseJSON(response, DecisionResult)
```

4.2 Algorithm 5': Multimodal Responder

Algorithm 8 Proposed `decide_and_respond()` — Accepts images

```
function LLM.DecideAndRespond(message, context, cases, images)
    system ← P_RESPOND_SYSTEM                                          ▷ Same prompt
    user_content ← []

    // Text part
    text_part ← "MESSAGE:" + message
    text_part ← text_part + "\n\nCONTEXT:" + context
    text_part ← text_part + "\n\nRETRIEVED CASES:" + JSON(cases)
    user_content ← user_content + [{"type" : "text", "text" : text_part}]

    // Images part (user's screenshots + KB evidence)
    for i, img_bytes in Enumerate(images) do
        if i < MAX_IMAGES_PER_REQUEST then          ▷ Limit to avoid token overflow
            b64 ← Base64Encode(img_bytes)
            mime ← DetectMimeType(img_bytes)
            user_content ← user_content + [{
                  "type" : "image_url",
                  "image_url" : {"url" : "data:" + mime + ";base64," + b64}
            }]

    response ← Gemini.ChatCompletion(system, user_content)
    return ParseJSON(response, RespondResult)
```

5 Database Schema Changes

5.1 Table: raw_messages

```sql
-- Add column to store image paths
ALTER TABLE raw_messages
ADD COLUMN image_paths_json TEXT DEFAULT '[]';

-- Example content:
-- ["attachments/abc123.png", "attachments/def456.jpg"]
```

Listing 1: Schema change for raw_messages

5.2 Table: cases

```sql
-- Add column to store evidence image paths
ALTER TABLE cases
ADD COLUMN evidence_image_paths_json TEXT DEFAULT '[]';

-- Example content:
-- ["attachments/abc123.png", "attachments/def456.jpg"]
```

Listing 2: Schema change for cases

6 Configuration Changes

```python
@dataclass(frozen=True)
class Settings:
    # ... existing fields ...

    # NEW: Multimodal settings
    max_images_per_gate: int = 3        # Max images for decide_consider
    max_images_per_respond: int = 5     # Max images for decide_and_respond
    max_kb_images_per_case: int = 2     # Max images from each KB case

    # Image size limits (to avoid token overflow)
    max_image_size_bytes: int = 5_000_000  # 5MB per image
    max_total_image_bytes: int = 20_000_000  # 20MB total per request
```

Listing 3: New settings in config.py

7 Prompt Changes

The prompts remain largely the same since Gemini naturally handles multimodal input. However, we can enhance them slightly:

```
P_DECISION_SYSTEM = """Determine if new message should be considered
for bot response.
Return ONLY JSON with keys:
- consider: boolean

consider=true only if:
- message asks for help or clarification, AND
- not trivial (greetings, "ok", emoji only), AND
- relates to group support context.

IMPORTANT: If message contains images (error screenshots, hardware
photos, wiring diagrams), consider their content when deciding.
Questions like "look at this" or "what's wrong here" with image
```

```
= consider=true.
"""

P_RESPOND_SYSTEM = """Decide whether to respond in group, prepare
response if yes.
Return ONLY JSON with keys:
- respond: boolean
- text: string (empty if respond=false)
- citations: array of short strings (e.g., ["case:123"])

Rules:
- respond=true only if you can answer using found cases and context.
- If uncertain, set respond=false (don't guess).
- Keep response short and to the point.
- Respond in UKRAINIAN.
- If responding, add 1-3 references to relevant cases.

IMPORTANT: If user provided images, analyze them carefully:
- Error screenshots: identify error code, program context
- Hardware photos: identify device type, visible issues
- Diagrams: understand wiring/connection scheme
Base response on what you SEE in images + KB case information.
"""
```

Listing 4: Enhanced prompts.py

8 Implementation Checklist

8.1 Phase 1: Fix Garbage Cases (P0)

- ☐ Add validation in `_handle_buffer_update()` to reject cases where `status="solved"` but `solution_summary` is empty
- ☐ Add logging for rejected cases
- ☐ Run cleanup script to remove existing garbage cases from Chroma
- ☐ Re-run eval to measure improvement

8.2 Phase 2: Database Schema (P3)

- ☐ Add `image_paths_json` column to `raw_messages`
- ☐ Add `evidence_image_paths_json` column to `cases`
- ☐ Update `insert_raw_message()` to store image paths
- ☐ Update `insert_case()` to store evidence image paths
- ☐ Update `get_raw_message()` to return image paths

8.3 Phase 3: Multimodal LLM Client (P1, P2)

- ☐ Update `LLMClient._json_call()` to accept optional `images` parameter
- ☐ Update `decide_consider()` signature to accept `images: list[bytes]`
- ☐ Update `decide_and_respond()` signature to accept `images: list[bytes]`
- ☐ Add image size validation and limiting
- ☐ Add MIME type detection for images

8.4 Phase 4: Worker Pipeline (P1, P2)

- ☐ Update `_handle_maybe_respond()` to load images from `msg.image_paths`
- ☐ Update `_handle_maybe_respond()` to load KB case evidence images
- ☐ Pass images to `decide_consider()`
- ☐ Pass images to `decide_and_respond()`
- ☐ Add configuration for image limits

8.5 Phase 5: Testing and Evaluation

- ☐ Update `run_streaming_eval.py` to test multimodal flow
- ☐ Add test cases with images
- ☐ Re-run full eval and compare pass rates
- ☐ Measure latency impact of image loading

9 Expected Improvements

Metric	Current	Expected
"answer" pass rate	8.7%	40–60%
"ignore" pass rate	87.1%	85–90%
Garbage cases in KB	43%	<5%
Overall pass rate	61.3%	70–80%

Table 2: Expected metric improvements

Note: The "answer" pass rate may not reach very high levels because many questions in the eval set ask about topics not covered in the KB. The multimodal fix helps the bot *understand* questions better, but it cannot answer questions for which no knowledge exists.

10 Risk Assessment

Risk	Severity	Mitigation
Increased latency from image loading	Medium	Limit image count and size; async loading
Token overflow with many images	High	Enforce `max_images_per_request` limit
Increased API costs	Low	Images processed by same Gemini call; minimal extra cost
Image storage growth	Low	Images already stored by Signal; we only store paths

Table 3: Risk assessment

11 Conclusion

The proposed changes address the two main issues identified in the evaluation:

1. **Garbage KB**: Fixed by validating that solved cases have non-empty solutions before storage.
2. **Missing visual context**: Fixed by passing raw images to Gemini at decision and response stages.

The approach maintains text-only embeddings for retrieval (avoiding the complexity of multimodal embeddings like Voyage AI), while leveraging Gemini's native multimodal capabilities for understanding user screenshots and evidence images.

This is a pragmatic "Option B" approach that delivers significant value with moderate implementation effort.