SupportBot Multimodal Implementation Report

Implementation Details, Evaluation Setup, and Examples

AI Agent (Cursor)

February 9, 2026

Abstract

This report documents the implementation of multimodal (text + images) support in SupportBot. It describes how image attachments are persisted, how images are passed into Gemini model calls via Google's OpenAI-compatible endpoint, and what evaluation scripts are included in the repository. Quantitative results depend on the availability of decrypted Signal history/attachments and API credentials; this document focuses on reproducible behavior verifiable from the codebase.

Contents

1 Executive Summary

1.1 Key Achievements

Capability	Evidence in codebase
Persist image attachment paths	`raw_messages.image_paths_json` (see `signal-bot/app/db/schema.py`, `signal-bot/app/db/schema_mysql.py`)
Image-to-text extraction at ingestion	`signal-bot/app/ingestion.py` calling `LLMClient.image_to_text_json()`
Pass images into gate/response LLM calls	`signal-bot/app/jobs/worker.py` passes images into `LLMClient.decide_consider()` and `LLMClient.decide_and_respond()`
Store evidence image paths on cases	`signal-bot/app/jobs/worker.py:_collect_evidence_image_paths()` and DB field `cases.evidence_image_paths_json`
Surface KB evidence images at response time	`signal-bot/app/jobs/worker.py` loads images from retrieved case metadata `evidence_image_paths`

Table 1: Multimodal capabilities implemented (verifiable in code)

1.2 Implementation Status

All priority items from the proposed fix have been implemented:

- ✓ **P0**: Reject cases without solution_summary (High impact)
- ✓ **P1**: Pass images to `decide_and_respond()` (High impact)
- ✓ **P2**: Pass images to `decide_consider()` (Medium impact)
- ✓ **P3**: Store image paths in `raw_messages` (Enables P1/P2)
- ✓ **P4**: Include images in KB case evidence (Medium impact)

2 Algorithms (Current Implementation)

This section presents pseudoalgorithms that mirror the current multimodal implementation in the codebase.

2.1 Algorithm 1: Multimodal Message Ingestion

Algorithm 1 Multimodal Message Ingestion — Preserves image paths for later use

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

  for $path$ **in** $image\_paths$ **do**
    $img\_path \leftarrow$ ResolveAttachmentPath($path, settings.signal\_bot\_storage$)
    $img\_path \leftarrow$ Resolve(img_path)
    if $\neg img\_path$.Exists() **then**
      Log.Warning("Attachment missing: {path}")
      continue
    end if

    $stored\_image\_paths$.Append(img_path) ▷ Store canonical path

    ▷ Extract text/observations for searchability
    $img\_bytes \leftarrow$ ReadFile(img_path)
    $extraction \leftarrow$ LLM.ImageToTextJSON($img\_bytes, context\_text$)
    $content\_text \leftarrow content\_text +$ "[image]" + JSON($extraction$)
    ▷ On extraction error, store placeholder JSON {observations: [], extracted_text: ""}
  end for

  ▷ Store image paths alongside text
  InsertRawMessage($msg\_id, group\_id, ts, sha256(sender)$[: 16],
    $content\_text, stored\_image\_paths, reply\_to$)

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

Notable behavior:

- Image paths are stored in the database for later multimodal calls
- Relative attachment paths are resolved against `SIGNAL_BOT_STORAGE`
- Image-to-text extraction output is appended to message text for searchability

2.2 Algorithm 2: Case Extraction with Validation

Algorithm 2 Case Extraction with Solution Validation — Reject solved cases without solutions

```
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:     end if
12:
13:     case ← LLM.MakeCase(extract.case_block)
14:     if ¬case.keep then
15:         SetBuffer(group_id, extract.buffer_new)
16:         return
17:     end if
18:
19:                                  ▷ Reject solved cases without solutions
20:     if case.status = "solved" ∧ case.solution_summary.Strip() = "" then
21:         Log.Warning("Rejecting solved case without solution_summary")
22:         SetBuffer(group_id, extract.buffer_new)
23:         return
24:     end if
25:
26:     case_id ← NewUUID()
27:
28:                          ▷ Collect image paths from evidence messages
29:     evidence_image_paths ← CollectEvidenceImages(case.evidence_ids)
30:
31:     InsertCase(case_id, group_id, case.∗, evidence_image_paths)
32:
33:     doc_text ← JoinLines(case.problem_title, case.problem_summary,
                      case.solution_summary, "tags: " + Join(case.tags))
34:     embedding ← LLM.Embed(doc_text)
35:
36:                          ▷ Store image paths in metadata for retrieval
37:     Chroma.Upsert(case_id, doc_text, embedding,
                  {group_id, status, evidence_ids, evidence_image_paths})
38:
39:     SetBuffer(group_id, extract.buffer_new)
40: end procedure
41:
42: procedure CollectEvidenceImages(evidence_ids)
43:     paths ← []
44:     for msg_id in evidence_ids do
45:         msg ← GetRawMessage(msg_id)
46:         if msg ≠ null then
47:             for p in msg.image_paths do
48:                 paths.Append(p)
49:             end for
50:         end if
51:     end for
```

Notable behavior:

- Solved cases must have non-empty solution summaries
- Evidence image paths collected from raw messages
- Image paths stored in vector DB metadata for later retrieval

2.3 Algorithm 3: Multimodal Response Pipeline

Algorithm 3 Multimodal Response Pipeline — Images at every decision point

procedure HANDLEMAYBERESPOND($group\_id, msg\_id$)
  $msg \leftarrow$ GETRAWMESSAGE(msg_id) ▷ Now includes image_paths
  $context \leftarrow$ GETLASTNMESSAGES($group\_id, n$)

  ▷ Load images from current message for gate
  $msg\_images \leftarrow$ LOADIMAGES($msg.image\_paths, max\_gate, budget$)

  $force \leftarrow$ MENTIONSBOT($msg.content\_text$)
  if $\neg force$ **then**
    ▷ Gate sees images
    $decision \leftarrow$ LLM.DECIDECONSIDER($msg.content\_text, context, msg\_images$)
    if $\neg decision.consider$ **then**
      return ▷ Ignore greeting/noise
    end if
  end if

  $query\_embedding \leftarrow$ LLM.EMBED($msg.content\_text$)
  $retrieved \leftarrow$ CHROMA.RETRIEVE($group\_id, query\_embedding, k$)

  ▷ Collect images from retrieved KB cases
  $kb\_paths \leftarrow []$
  for $item$ **in** $retrieved$ **do**
    $paths \leftarrow item.metadata.evidence\_image\_paths$
    kb_paths.EXTEND($paths[: max\_per\_case]$)
  end for
  $kb\_paths \leftarrow kb\_paths[: max\_total\_kb]$

  ▷ Load KB images (respecting budget after msg images)
  $remaining\_budget \leftarrow$ MAX($budget -$ TOTALSIZE(msg_images), 0)
  $kb\_images \leftarrow$ LOADIMAGES($kb\_paths, max\_respond, remaining\_budget$)

  $all\_images \leftarrow msg\_images + kb\_images$
  $all\_images \leftarrow all\_images[: max\_images\_per\_respond]$ ▷ Final cap

  $cases\_json \leftarrow$ JSON($retrieved$)

  ▷ Responder sees all images
  $resp \leftarrow$ LLM.DECIDEANDRESPOND($msg.content\_text, context,$
    $cases\_json, all\_images$)

  if $resp.respond$ **then**
    SIGNAL.SEND($group\_id, resp.text$)
  end if
end procedure

procedure LOADIMAGES($paths, max\_count, budget\_bytes$)
  $images \leftarrow []$
  $total \leftarrow 0$
  for p **in** $paths$ **do**
    if $|images| \geq max\_count$ **then**
      break
    end if
    $data \leftarrow$ READFILE(p)

Notable behavior:

- **P2**: Gate stage receives images from user message
- **P1**: Responder receives images from both user message and KB evidence
- **P4**: Evidence images retrieved from case metadata
- Image budgets cap multimodal payload size (`MAX_IMAGE_SIZE_BYTES` and `MAX_TOTAL_IMAGE_BYTES`)

3 Examples (Illustrative)

This section provides illustrative, synthetic examples to show how messages are transformed into structured cases and how images flow through the pipeline. Message IDs, timestamps, similarity values, and file paths shown below are placeholders and will differ in real runs.

3.1 Example 1: Compatibility Question (synthetic)

3.1.1 Raw Messages (Input)

```
User:
Is device MODEL_X supported? Which target should I select when flashing?

Support:
Yes. Use target TARGET_Y. The tooling will detect it as TARGET_Y.
```

3.1.2 Extracted Case Block

```
CASE BLOCK:
problem: Device compatibility and flashing target selection
evidence: [msg_id_1, msg_id_2]
status: solved
```

3.1.3 Structured Case (Output)

Field	Value
`case_id`	`c4f2a891-...`
`status`	`solved`
`problem_title`	Support for device MODEL_X
`problem_summary`	User asks whether MODEL_X is supported and what target to select when flashing.
`solution_summary`	Support confirms compatibility and recommends selecting target TARGET_Y in the flashing tool.
`tags`	compatibility, firmware, flashing, target selection
`evidence_ids`	[msg_id_1, msg_id_2]
`evidence_image_paths`	[] (no images in this case)

Table 2: Structured case with metadata

3.1.4 Embedding & Storage

- **Document text**: Built from title + summaries + tags (see `signal-bot/app/jobs/worker.py`)
- **Embedding**: Vector produced by the configured `EMBEDDING_MODEL`
- **Vector DB**: Stored in ChromaDB with metadata: `{group_id, status, evidence_ids, evidence_image_paths}`

3.2 Example 2: Multimodal Message with Screenshot (synthetic)

3.2.1 Raw Messages (Input)

```
User:
Please help. I see an error when trying to complete an action. What does it mean?
[image: <attachment.png>]

Support:
Which mode/step are you in when the error appears?

User:
It happens in MODE\_A. In MODE\_B it works normally.

Support:
Please share logs/config. The screenshot may include an error code we can diagnose.
```

3.2.2 Structured Case with Image Paths

Field	Value
`problem_title`	Error shown in screenshot during MODE_A
`problem_summary`	User cannot complete an action in MODE_A; screenshot contains an error message/code. MODE_B works.
`solution_summary`	Gather logs/config and troubleshoot based on the exact error text visible in the screenshot and retrieved evidence cases.
`evidence_image_paths`	`["<abs_path_to_attachment.png>"]`

3.2.3 How Images Are Used

At ingestion:

- Image extracted to text: `{observations: ["..."], extracted_text: "..."}`
- Image path stored: `<abs_path_to_attachment.png>`

At retrieval (when user asks similar question):

1. User query: "What does this screenshot error mean?"
2. System retrieves relevant case(s) via semantic similarity search
3. Loads evidence image(s) from disk (bounded by size/budget limits)
4. Passes image(s) + retrieved case text to the LLM for response generation

4 Solved Cases: Retrieval Introspection

This section demonstrates how the bot retrieves and reasons about cases when answering user questions, with full introspection into the retrieval pipeline.

4.1 Example Query 1: Gimbal Control Issue

4.1.1 User Question

```
User: Need to control FEATURE\_X but it doesn't work on this configuration.
It looks like DEVICE\_Y lacks MODE\_Z. RC passthrough alone is not enough.
```

4.1.2 Stage 1: Semantic Search

Query embedding: Generated from user question
Search parameters:

- `group_id`: `<group_id>`
- `k`: `RETRIEVE_TOP_K` (default: 5)
- `embedding_model`: `EMBEDDING_MODEL` (configurable)

Retrieved cases (ranked by similarity score):

Rank	Case Title
1	FEATURE_X control troubleshooting
2	Output/servo mode configuration
3	Missing MODE_Z support on DEVICE_Y
4	RC passthrough configuration
5	Firmware option enabling FEATURE_X

Table 3: Top-5 retrieved cases

4.1.3 Stage 2: Image Loading

For each retrieved case:

- Case 1: `evidence_image_paths` = `[]` (no images)
- Case 2: `evidence_image_paths` = `[]` (no images)
- Case 3: `evidence_image_paths` = `[]` (no images)
- Case 4: `evidence_image_paths` = `[]` (no images)
- Case 5: `evidence_image_paths` = `[]` (no images)

Total images loaded: `<n>`
Total budget used: `<bytes>` / `MAX_TOTAL_IMAGE_BYTES`

4.1.4 Stage 3: LLM Decision

Input to LLM:

- User message (text)
- Context: last 40 messages from group
- Retrieved cases: JSON with top-5 cases
- Images: 0 message images + 0 KB images = 0 total

LLM output:

```
{
  "respond": true,
  "text": "<answer based on retrieved case evidence>",
  "citations": ["case:<case_id>"]
}
```

4.2 Example Query 2: Configuration Parameter Question

4.2.1 User Question

```
User: Good day. Analyzing parameter "PARAM\_X": Value: V. If the system error
exceeds this threshold, a failsafe will trigger. Is this too low? What are the recommendations?
```

4.2.2 Retrieval Results

Rank	Case Title	Status
1	PARAM_X configuration	solved
2	Failsafe parameters	solved
3	Sensor/estimation troubleshooting	solved

Top case evidence:

- **Problem**: User asks about a configuration parameter and whether the current value is appropriate
- **Solution**: `<solution summary from retrieved case evidence>`
- **Evidence IDs**: [msg_id_42, msg_id_43, msg_id_44]
- **Images**: No images in evidence

4.2.3 Bot Response

```
<answer based on retrieved case evidence>

Ref: case:<case_id>...
```

4.3 Example Query 3: Multimodal Response (with Screenshot)

4.3.1 User Question with Image

```
User: Look at what the ground control app shows. Don't understand what
kind of error this is. [image: <attachment.png>]
```

4.3.2 Image Processing Pipeline

Step 1: Message ingestion

- Image path stored in DB: `<abs_path_to_attachment.png>`
- Image-to-text JSON appended to `content_text`: `{"observations": ["..."], "extracted_text": "..."}`

Step 2: Gate decision (`decide_consider`)

- Message text: "Look at what the ground control app shows..."
- **Images loaded:** up to `MAX_IMAGES_PER_GATE` (subject to size/budget limits)
- LLM sees: text + the attached image(s)
- Decision: `consider=<true/false>`

Step 3: Semantic retrieval

- Query embedding from: `<user message text>`
- Top retrieved case: `<case title>`
- Case may include `evidence_image_paths`: `["<abs_path_to_evidence.png>"]`

Step 4: Response generation (`decide_and_respond`)

- **Message images:** `<n>`
- **KB images:** `<n>`
- **Total images:** capped by `MAX_IMAGES_PER_RESPOND`
- **Total bytes:** capped by `MAX_TOTAL_IMAGE_BYTES`
- LLM sees: user image(s) + retrieved case text (+ optional KB evidence image(s))

4.3.3 Bot Response

```
<answer based on retrieved case evidence>

Ref: case:<case_id>...
```

Why multimodal can help (conceptually):

- Without images: the model only sees text, which may be ambiguous
- With images: the model can directly see UI text/error codes and other visual context
- With KB evidence images: the model can compare the user's screenshot to historical evidence

5 Evaluation and Reproducibility

This repository contains tests and evaluation scripts. Most evaluation outputs are written under `test/data/` and are intentionally gitignored; quantitative metrics therefore depend on the local environment, available decrypted Signal history/attachments, and API credentials.

5.1 Unit tests (offline)

- Run: `pytest -v`
- Core coverage includes ingestion, buffer/case extraction, Chroma integration, and response gating.

5.2 LLM-backed quality evaluation (requires `GOOGLE_API_KEY`)

- Tests: `test/test_quality_eval.py`
- The judge uses Gemini via Google's OpenAI-compatible endpoint.

5.3 Real-data evaluation (optional; requires decrypted Signal history)

- Mine cases: `python test/mine_real_cases.py`
- Run eval: `python test/run_real_quality_eval.py`
- Streaming eval: `python test/run_streaming_eval.py`
- Image-to-text demo: `python test/run_image_to_text_demo.py`

5.4 Notes on Signal Desktop encryption

Signal Desktop backups may require Windows DPAPI decryption under the same Windows user account that created the backup (see `test/results.md` for a documented example).

5.5 Embedding model note

- Application default: `EMBEDDING_MODEL=text-embedding-004` (see `signal-bot/app/config.py`).
- The real-eval script may override to `gemini-embedding-001` for compatibility with the OpenAI-style Gemini embeddings endpoint (see `test/run_real_quality_eval.py`).

6 Configuration and Limits

6.1 Multimodal Settings

Parameter	Value	Purpose
`MAX_IMAGES_PER_GATE`	3	Limit images sent to gate decision
`MAX_IMAGES_PER_RESPOND`	5	Limit total images in response call
`MAX_KB_IMAGES_PER_CASE`	2	Limit evidence images per retrieved case
`MAX_IMAGE_SIZE_BYTES`	5,000,000	Skip images $>$ 5,000,000 bytes
`MAX_TOTAL_IMAGE_BYTES`	20,000,000	Total budget per response (20,000,000 bytes)

Table 4: Image budget limits (caps multimodal payload size)

6.2 Cost considerations

The main cost drivers are LLM calls during ingestion and response:

- **Ingestion**: optional image-to-text extraction per attachment (`image_to_text_json`)
- **Retrieval**: embeddings for case documents and user queries (`embed`)
- **Response**: gate (`decide_consider`) and responder (`decide_and_respond`) chat calls, optionally with images

Actual costs depend on model selection and provider pricing. The implementation enforces strict caps on image count and total bytes to bound multimodal payload size (Table 4).

7 Conclusion

This implementation adds end-to-end multimodal plumbing:

1. **Reject low-quality cases** (P0): Reject `status=solved` cases without `solution_summary`
2. **Preserve image references** (P3): Store attachment paths in `raw_messages.image_paths_json`
3. **Use images in decisions and responses** (P1, P2): Pass images into `decide_consider` and `decide_and_respond`
4. **Carry evidence images through retrieval** (P4): Store and retrieve `evidence_image_paths` via Chroma metadata

Measuring impact: Use the scripts in Section 5 to run evaluations in an environment with decrypted data and valid API credentials.

Next steps:

- Deploy to production and monitor real-world performance
- Gather user feedback on response quality
- Fine-tune retrieval thresholds based on precision/recall metrics
- Consider adding image captioning for better searchability