ChatGPT

Enhanced Email Answer Quality Strategy Plan

Overview

This plan builds upon the existing email **agentic retrieval strategy** to further maximize **answer quality** when responding to queries with email data. It identifies and addresses weaknesses in the current approach – such as incomplete thread context, irrelevant or redundant content, and limited intent handling – and introduces new components and algorithms to ensure answers are **complete, accurate, and contextually rich**. The focus is on enhancing how email content is retrieved, assembled, and utilized by the LLM so that responses reflect the full relevant conversation with minimal noise. Key improvements include more sophisticated context assembly (with thread reconstruction and noise reduction), dynamic retrieval techniques for complex queries, intent-aware answer formulation, and post-processing steps like summarization and verification. By grounding these enhancements in the project's existing architecture, we ensure a feasible upgrade path that tightly integrates with current components.

Key Enhancements

- **Robust Thread Reconstruction & Deduplication:** Improve email thread assembly by capturing *entire conversations* while removing redundant quoted text and signatures. This ensures the LLM sees the full dialogue in chronological order without confusion from duplicate content [1] [2]. The result is more coherent answers for thread-related queries.
- **Dynamic Multi-Aspect Retrieval:** Extend the strategy router to handle **multi-faceted intents**. For queries combining factors (e.g. *"What did Alice say about X last week?"*), apply **combined filtering** (sender + temporal) rather than a single static strategy. This multi-aspect approach prevents missing relevant context due to rigid one-strategy selection.
- **LLM-Assisted Intent Detection:** Augment the pattern-based intent classifier with a lightweight LLM step for ambiguous cases and to capture nuanced intents. This hybrid method boosts accuracy for complex queries and ensures the correct strategy is chosen even when user phrasing doesn't match predefined regex patterns [3] [4]. High-confidence pattern matches remain fast, while uncertain cases get an LLM "second opinion" for reliability.
- **Advanced Context Assembly & Summarization:** Introduce a **Context Assembler** module that not only collates retrieved email chunks, but also **compresses or summarizes** lengthy threads when needed. By using discourse cues (e.g. reply indentation or quoted text markers), the assembler can strip boilerplate and even summarize parts of a long thread (via an LLM or heuristic) to fit within token limits. This layered summarization ensures that even very long discussions can be provided to the LLM in digestible form, preserving critical points for answer generation.
- **Enhanced Retrieval Ranking and Noise Filtering:** Upgrade semantic search with secondary re-ranking (e.g. a cross-encoder or heuristic scoring) to prioritize the most relevant email passages. Filter out irrelevant emails and system-generated noise (newsletters, automated replies) by leveraging metadata (sender domains, folders) to avoid distracting the LLM. By feeding the LLM only highly relevant, human-authored content, we improve answer precision.
- **Intent-Aligned Answer Prompting:** Tailor the final LLM prompt according to the detected intent. For example, for a `thread_summary` intent, explicitly instruct the LLM to **summarize the**

conversation with key points; for `action_items`, prompt it to **extract and list tasks or deadlines**; for `sender_query`, ask it to focus on **what that person said** (potentially quoting them). Aligning the answer format with the query intent yields more directly useful and higher-quality answers.

- **Answer Verification & Contradiction Checking:** As a post-processing layer, incorporate an **Answer Validator** step to catch errors or inconsistencies. After the LLM generates an answer, a smaller verification model or logic can check the answer against the retrieved sources for unsupported claims or contradictory information. For instance, if two emails gave different figures for a budget, the system can detect this and either reconcile it or flag the uncertainty. This layer helps ensure **factual correctness** and increases trust in the answers.
- **Expanded Specialized Strategies:** Implement the planned **Action Item Extraction** and **Decision Tracking** strategies fully. Leverage either rule-based detectors or LLMs to scan emails for task assignments, deadlines (for `action_items` intent) and for conclusions or approvals (for `decision_tracking` intent). These strategies will output concise lists of action items or decisions, directly answering those queries. This closes a gap in the current plan where these intents were identified but not yet operationalized [5], thereby improving answer quality for those cases.

Improved Architecture

The enhanced architecture retains the modular **agentic orchestration** design [6] but adds new components and data flows to enrich the context for the LLM and validate its outputs. The high-level flow is now:

```
┌───────────────────────────────────────────────────────────┐
│             Email Orchestrator Agent (Enhanced)           │
│                                                           │
│   ┌────────────┐    ┌────────────────┐                    │
│   │Intent      │    │Strategy        │                    │
│   │Detector 🔍│──→│Selector 🗺️│                 │
│   └─────▲──────┘    └───────┬────────┘                    │
│         │  (LLM fallback)   │           Combined?         │
│         └────────no/yes─────┘              │              │
│                             ↓              ▼             │
│              ┌─────────────────────────────┐             │
│              │ Retrieval Strategy Router │             │
│              └───────────┬───────┬─────────┘             │
│                  (Thread│Temporal│Sender + others)        │
│                         ▼    ▼       ▼                   │
│         ┌─────────┐ ┌────────┐ ┌─────────┐ ┌──────────┐  │
│         │Thread   ││Temporal││Sender   ││Action/Decision│…   │
│         │Retriever││Retriever││Retriever││Extractors   │   │
│         └───┬─────┘ └────┬───┘ └───┬─────┘ └───┬────────┘ │
│             │ Filter +   │       │         │ (Can combine) │
│             │  Group     │       │         │              │
│             ▼            ▼       ▼         ▼              │
│       ┌─────────────────────────────────────┐            │
│       │   **Context Assembler & Refiner**   │           │
│       │   - merge threads chronologically   │           │
│       │   - remove duplicates/quotes        │           │
│       │   - summarize if too large          │           │
│       └─────────────────┬─────────────────┘           │
│                         ↓                               │
│                Assembled Context                      │
│                         │                               │
│                         ↓                               │
│       ┌─────────────────────────────────────┐            │
│       │      LLM Answer Generator         │          │
│       └───────────────┬───────────────────┘           │
│                       │ (Answer + citations)           │
│                       ↓                                 │
│         ┌─────────────────────────────────┐           │
│         │   Answer Validator (QA check)   │           │
│         └─────────────────────────────────┘           │
│                       │                                 │
│                Final Answer                             │
└──────────────────────────────────────────────────────┘
```

What's New: The **Intent Detector** now has an optional LLM-based classifier to handle tricky queries. The **Strategy Selector** can combine multiple strategies or apply additional filters as needed (for example, it might invoke the SenderRetriever and then apply TemporalRetriever on its results). New specialized agents for Action and Decision extraction are integrated into the router for their respective intents. A beefed-up **Context Assembler & Refiner** module takes over responsibility for cleaning and organizing the retrieved emails into a cohesive context (performing tasks like deduplication of quoted text, chronological merging of threads, and on-the-fly summarization of lengthy content). After the context is passed to the **LLM Answer Generator**, an optional **Answer Validator** uses either an automated script or a secondary LLM pass to verify the answer against the context, checking for completeness and consistency before finalizing the response.

This improved architecture ensures that the LLM is fed with *exactly the information it needs* in an optimized form, and that its output is vetted, thereby significantly boosting the reliability and quality of answers derived from email data.

Component Changes

1. Intent Detection: The `EmailIntentDetector` gains an LLM-powered helper method for classification. The system will first attempt regex/pattern matching as before; if the top intent confidence is below a threshold (e.g. 0.6), it will call a small LLM (like GPT-3.5 or a local model) to classify the query [3] [4]. This increases accuracy for edge cases or complex wording. We also extend the detector to identify *multiple intent signals* in a single query. For instance, if a query mentions both a person and a time frame, the detector's output `metadata` can include both `"sender": X` and `"time_range": Y`. The detector might tag a primary intent (say `sender_query`) but also note secondary cues, enabling downstream use of combined strategies.

2. Strategy Selector: The `EmailStrategySelector` logic is upgraded to handle combined intents. Instead of mapping one intent to one strategy blindly, it can now produce **composite strategies**. For example, if `primary_intent` is `sender_query` but metadata also has a `time_range`, the selector could choose a new combined mode (e.g. `"sender+temporal_retrieval"`). Internally, this might call the SenderRetriever and then filter or rank results by date. The selection function will also route any unrecognized or low-confidence intents to a safe default (standard semantic retrieval) as before [7]. The strategy map is expanded to include the new specialized strategies: `"action_items"` -> `action_extraction` and `"decision_tracking"` -> `decision_extraction` (both implemented as described below).

3. Retrieval Agents:
- **ThreadRetriever:** Augmented to ensure **complete thread capture**. It will retrieve all emails belonging to top relevant threads as designed [8], then **eliminate duplicate content** stemming from quoted replies. This can be achieved by detecting common quote prefixes (`>` or email headers in text) or using known email threading libraries to isolate new content in each message. The output will be a sequence of unique email segments in order. This yields a full conversation without repeated text, directly improving coherence in the final answer.
- **TemporalRetriever:** Extended to support flexible time expressions (last week, yesterday, between X and Y). We'll implement the `parse_time_range` function to handle common phrases and perhaps integrate a library for robust date parsing. Additionally, if combined with other filters (like sender), it will intersect the conditions (e.g., filter by sender then date). The result set is sorted chronologically (newest first or as needed) to emphasize recent information [9].
- **SenderRetriever:** Improved with fuzzy matching of names (to handle nicknames or various email name formats) and possibly contact alias resolution. If the query provides only a first name and multiple contacts share it, we might use additional clues (perhaps the query topic or most frequent correspondent) to pick the right person's emails. This retriever can also accept an optional date filter from the strategy selector for combined queries.
- **Adaptive Retriever:** The standard semantic search (now called **AdaptiveRetriever**) will incorporate an intelligent **K tuning**. While the current plan sets K=15 for emails [10], we will make this dynamic: for very broad queries or when other strategies are not triggered, the system might retrieve more chunks (e.g., K=20) but then rely on re-ranking and context assembly to trim irrelevant pieces. Conversely, if a specific question is asked, a smaller K might suffice to avoid noise. This adaptivity will be guided by either intent (as before) or even the estimated relevance distribution (e.g., if top scores have a sharp drop-off, don't include low-scoring chunks).

- **ActionItemExtractor (New):** A specialized agent that scans emails for task-related content. This can be implemented by running a focused LLM prompt on the retrieved emails (or entire thread) asking for a list of action items and deadlines. Alternatively, we can implement a rule-based parser for patterns like "**TODO:**" or verbs like "need to" near dates. The extractor returns a summary or list of actions, which the orchestrator will pass as context (or directly as an answer draft) to ensure the final answer cleanly presents the tasks.
- **DecisionExtractor (New):** Similarly, this agent will identify statements of decisions or conclusions. Using an LLM, we can prompt it to find sentences where something was decided or agreed upon, or use regex for phrases ("decided that", "we will", "the conclusion is"). The output is a concise statement of the decision(s) made in the relevant emails. Both new extractors fulfill the `action_items` and `decision_tracking` intents with high-precision content, rather than relying on generic semantic search.

4. Context Assembler & Refiner: This is a critical new component that takes all retrieved chunks and **constructs a well-organized, cleaned context** before the LLM sees it. Its responsibilities:
- **Chronological Merge:** Ensure emails that are part of the same thread or time-sequenced are ordered properly (oldest to newest for a summary context, or newest first if answering “recent” queries).
- **Source Attributions:** (Optional) Insert brief headers or metadata tags before each email chunk, e.g. "**Sender: Alice (Jan 5, 2024):** ...email snippet...". This gives the LLM clarity on who said what and when, improving the quality of answers for questions about who said something or the timeline of events.
- **Redundancy Removal:** Strip out quoted previous messages and repetitive email signatures/disclaimers using known patterns. This significantly reduces noise and token waste. The assembler might keep only the unique content from each email. If an entire email is just a quoted reply of a previous one, it can be skipped.
- **Summarization & Compression:** If the assembled context is too large (e.g., a thread of dozens of long emails), invoke a summarization step. This could mean summarizing less relevant parts of the thread or truncating low-importance details. For example, the assembler might replace a series of less relevant emails with a note like "[... 3 intermediate responses omitted ...]" plus a short summary of them, focusing the LLM on the key parts of the discussion. This keeps context within token limits while preserving essential information.
- **Final Context Packaging:** The refined context is packaged for the LLM, possibly with an outline or key points first (to give the LLM an overview). By performing these steps, the assembler ensures the LLM’s input is *both comprehensive and concise*, directly boosting answer relevance and readability.

5. LLM Answer Generator: While the underlying LLM model remains the same, we improve how we **prompt** it. The system will choose a prompt template suited to the intent – for instance, for a thread summary it might prepend: "*You are an assistant summarizing an email thread.*" and instruct it to produce a structured summary. For factual lookups, it might encourage short, specific answers quoting the source if needed. For action items or decisions, it may prompt the LLM to list them clearly. By aligning the prompt with the query type, we guide the model to produce outputs that better satisfy the user’s request. Additionally, we ensure the prompt explicitly reminds the LLM to use **only the provided emails** for information, minimizing any risk of hallucination.

6. Answer Validator (Quality Check): After the LLM generates an answer, a new optional component will perform a quality check. This could be a simple script that verifies required elements are present (e.g., if the query was for action items, ensure the answer contains a list or bullets), or a more advanced check using an LLM: e.g., ask a second instance of the model "*Is every statement in this answer supported by the emails?*" or "*Do these emails contain contradictory information on the question?*". If the validator finds an issue (unsupported claim or conflict), the system can either:
- Automatically revise the answer (for example, by prompting the LLM to resolve the discrepancy or by appending a clarification), **or**
- Tag the answer with a caution or request for clarification.

In many cases, a straightforward check is enough – for example, ensuring that if multiple different answers exist in emails (like two different dates proposed for a meeting), the final answer notes both rather than arbitrarily picking one. This validation step acts as a safeguard to further improve factual accuracy and completeness of answers.

Implementation Roadmap

To integrate these enhancements into the existing project, we propose a phased implementation approach:

Phase 5: Enhanced Retrieval & Context (Week 1-2)
- **Intent Detector Upgrade:** Implement the LLM fallback in `EmailIntentDetector` and allow multi-metadata output. Add unit tests for queries with mixed intents and ambiguous phrasing.
- **Strategy Selector & Router:** Update `EmailStrategySelector` to support combined strategies. Introduce any new strategy keys (e.g., `"sender+temporal_retrieval"`). Adjust router to handle routing to multiple retrievers sequentially or in combination.
- **Thread & Sender Retriever Updates:** Modify `ThreadRetriever` to perform content deduplication (e.g., using regex to remove quoted text) and verify via tests that a reconstructed thread contains only unique content. Enhance `SenderRetriever` with fuzzy matching logic and test with partial names and email address variations.
- **Context Assembler Module:** Create the `ContextAssembler` class/module. Implement functions for sorting emails, labeling metadata, stripping quotes/signatures, and merging content. Write unit tests feeding it synthetic threads with known quoted text to ensure it cleans correctly (e.g., an email reply that includes the previous email should result in only one copy of that text in final output).
- **Adaptive K & Re-ranking:** Tune the retrieval pipeline so that after initial semantic search results are retrieved, a secondary ranking step (if using a library or simple vector dot-product re-sorting) is applied. If available, integrate a cross-encoder model for re-ranking as an experimental feature. Confirm that for a set of sample queries, the top chunks after re-ranking better align with query intent than before.

Phase 6: Intent-Aligned Answering (Week 3)
- **Prompt Template Design:** For each intent type, create a specialized prompt template or instructions. For example, define how to ask for summaries vs. lists vs. direct answers. Implement logic in the answer generation step to choose the prompt format based on `intent`.
- **Action/Decision Strategy Implementation:** Build the `ActionItemExtractor` and `DecisionExtractor`. Likely start with an LLM-based approach: craft prompts that, given a set of emails or a thread, return the list of action items or decisions. Alternatively, implement a simpler regex or keyword-based extraction as a first pass. Test these on known email examples (maybe create sample emails with obvious tasks/decisions to see if extraction works). Integrate these into the orchestrator so that if those intents are detected, the system uses these extractors to produce context (which might actually be the final answer content or part of it).
- **Answer Validator Prototype:** Develop a basic validation function. Initially, this could be non-LLM: for instance, check if the answer contains certain keywords expected for the intent (e.g., if the query was "What are the action items", ensure the word "Action" or bullet points are present; if the query asked for a person's statement, ensure that person's name is referenced in answer). Then, experiment with an LLM-based validator for a couple of scenarios (contradictory facts, unsupported info) using a small model or GPT-3.5 if available. Define the conditions under which the answer would be flagged or auto-corrected. Write a few integration tests where we intentionally feed contradictory context to ensure the validator catches it.

Phase 7: Integration & Performance Tuning (Week 4)
- **Integration Testing:** Run end-to-end tests with the full pipeline on realistic email data queries. For example, queries like *“Summarize the Q3 planning thread”*, *“What did Bob say about the hiring freeze last month?”*, *“List any action items from the migration emails”*, etc., and verify the answers are of high quality (comprehensive, correct, well-formatted). Compare these answers to those from the previous baseline system to quantify improvements (e.g., higher relevance, no missed key emails, no hallucinated info).
- **Performance Optimization:** Evaluate the speed impact of new components, especially the LLM-based intent detection and answer validation. If the LLM intent step is too slow for each query, consider caching frequent patterns or using a smaller model. Similarly, ensure that removing duplicates and summarizing

doesn’t introduce significant lag; optimize by doing heavy text processing (like quote stripping) offline during indexing if possible, or by using efficient regex.
- **User Experience Updates:** Update the UI or logs to expose the new capabilities – for instance, displaying which strategy was used (thread, temporal, etc.) and possibly any notes (e.g., “Multiple relevant threads found, combined in answer” or “Conflicting data detected, answer includes both possibilities”). This transparency can help in user trust and in debugging. Also, gather user feedback on the answer quality explicitly to further fine-tune the system.

By following this roadmap, we will incrementally roll out the enhancements, ensuring at each step that the system remains stable and that each change measurably improves the quality of answers. Throughout, we leverage the existing project structure (agents, retrievers, orchestrator) to incorporate these upgrades, making the **Email RAG module** far more adept at handling the complexity of email data. The end result will be an email QA system that delivers **high-quality, context-aware answers** with confidence, even in the face of long threads, myriad senders, and complex queries.

---

1 2 3 4 5 6 7 8 9 10 EMAIL_AGENTIC_STRATEGY_PLAN.md
file://file_0000000040d071f4892fe289670dbc85