Problem in Simple Words

Design and implement a personal AI companion that can ingest multi-modal user data (audio, documents, web pages, text; images supported via searchable metadata) and answer questions via a natural language chat interface. The system must support temporal questions (e.g., “last Tuesday”, “last month”), provide fast retrieval, and return grounded answers with citations.

REQUIREMENTS:

Functional requirements

Multi-modal ingestion

- Audio: accept .mp3/.m4a, transcribe to text, preserve timestamp spans
- Documents: accept .pdf and .md, extract text + metadata, preserve page/section pointers
- Web content: given a URL, fetch and extract main content + metadata
- Plain text: accept raw notes
- Images: propose method to store images and make them searchable via associated text/metadata

Information retrieval & Q&A

- Retrieve relevant context from user’s knowledge base using a justified strategy
- Use an LLM to synthesize an answer from retrieved context
- Return citations that point back to exact sources (page/time/url)

Temporal querying

- Associate timestamps with all ingested data
- Retrieval must use timestamps to answer time-based questions

Non-functional requirements

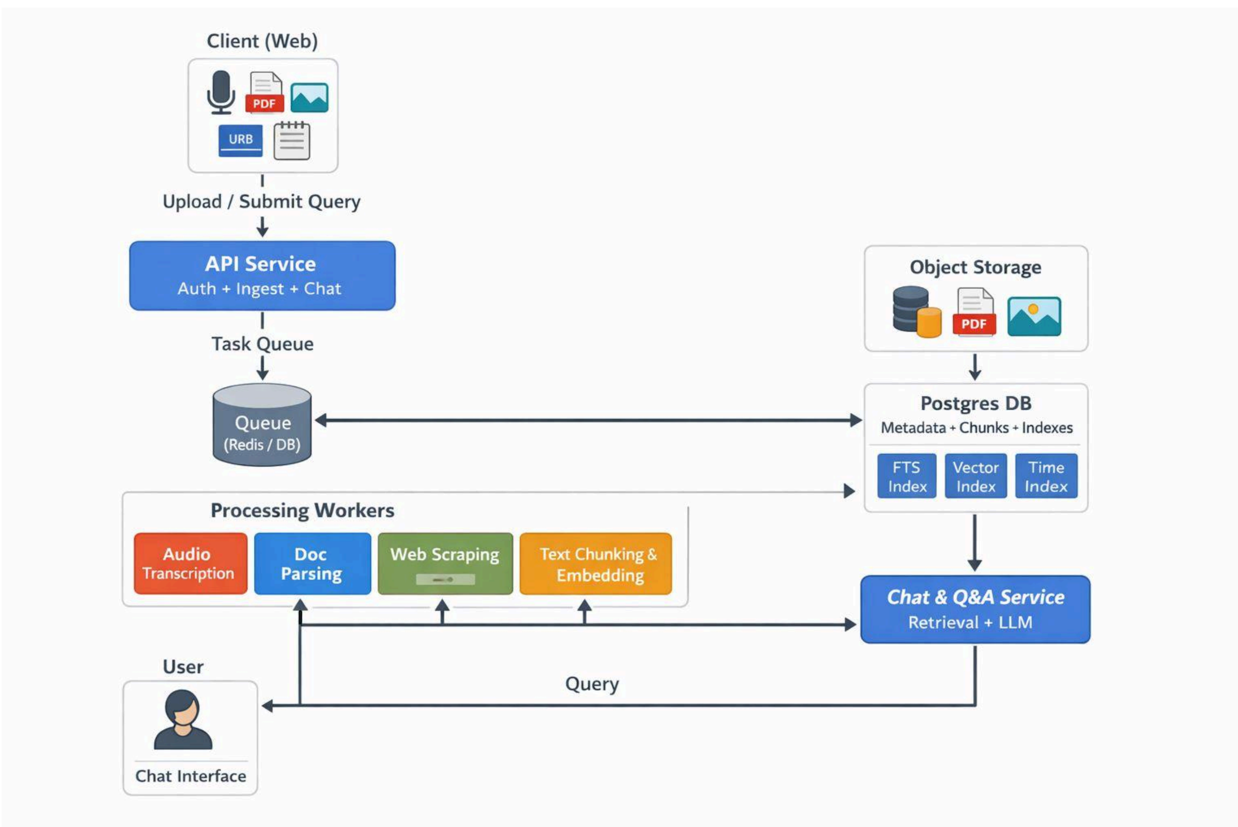
- Low-latency retrieval and good relevance
- Scales to thousands of documents per user
- Privacy by design (per-user isolation)
- Reliable ingestion (async processing, retries, idempotency)

High-level design

Use a “single source of truth + multiple indexes” design:

- Object Storage: raw artifacts (audio/pdf/images) and optional derived files
- Postgres: system of record for metadata + chunks + timestamps
- Indexes in Postgres:
- FTS (keyword)
- pgvector (semantic embeddings)
- time indexes (temporal filtering)
- Async workers handle ingestion pipelines
- Chat/Q&A service runs retrieval + LLM synthesis
- Frontend provides ingestion + chat with streaming

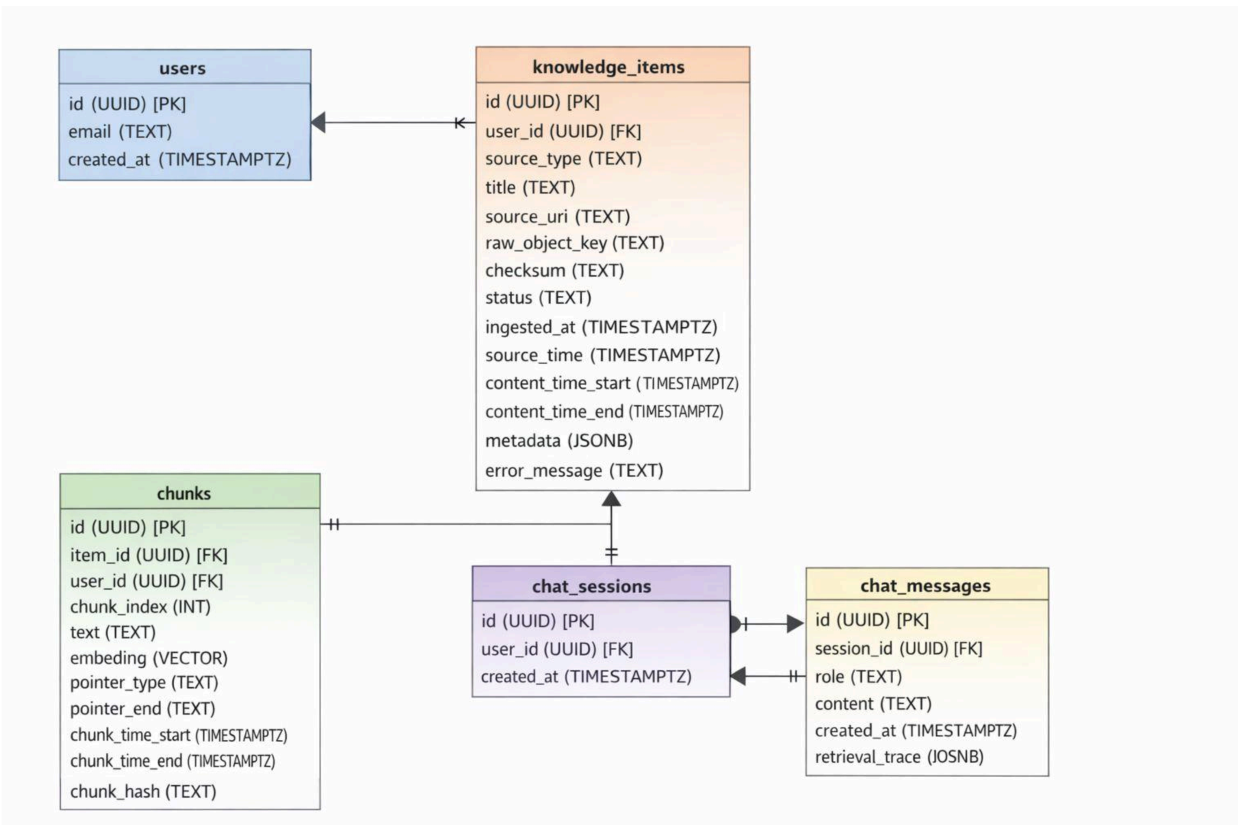
Architecture diagram



Data model and storage (schema + indexing)

Why this model is optimal

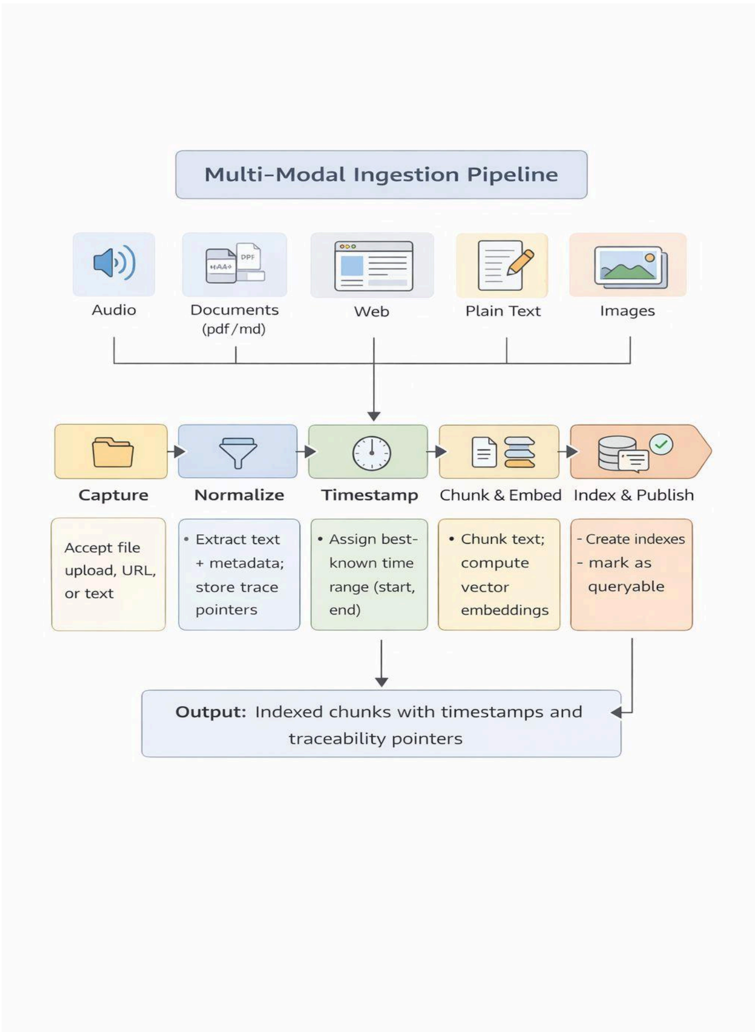
- Traceability: every chunk stores a pointer back to source (page/time/url)
- Time-first: timestamps are first-class fields used in retrieval
- Hybrid search: keyword + semantic + time filters from one DB
- Scalable per user: indexes and filtering always scoped by user_id



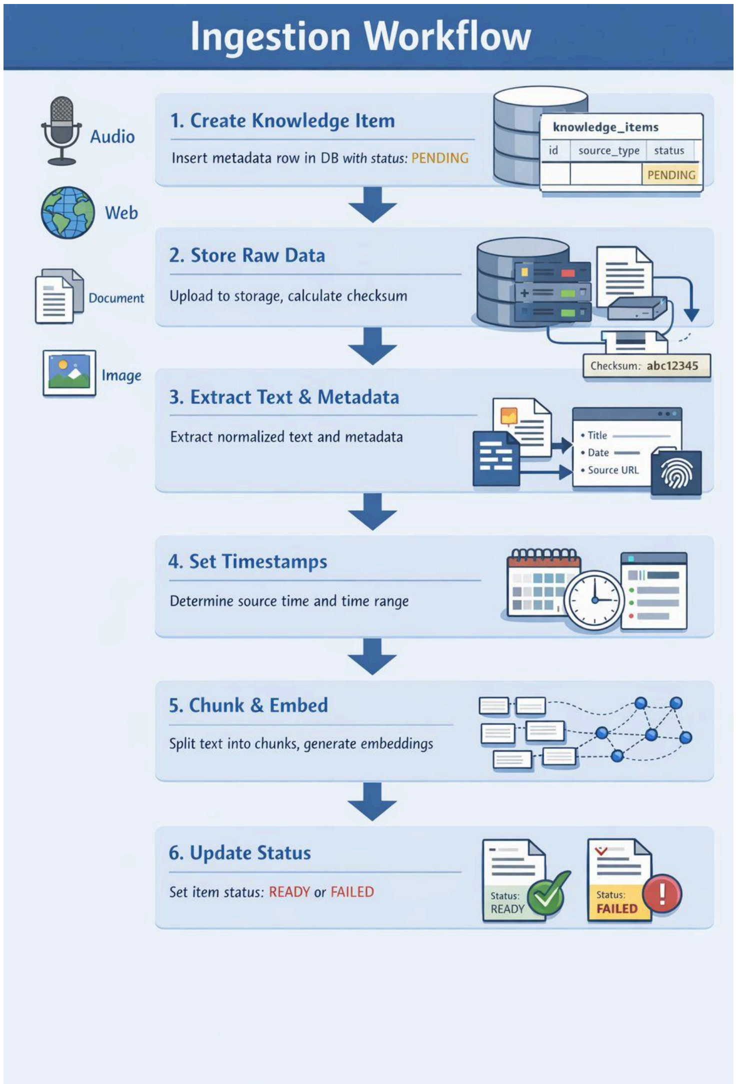
Chunking strategy

- Prefer natural boundaries first:
- PDF pages / markdown headings / web sections / audio segments
- Then apply token-window chunking:
- target ~400–700 tokens with overlap
- Store chunk_hash so retries don't duplicate chunks.

Multi-modal ingestion pipeline



COMMON MODAL WORKFLOW:



Modality-specific details

Audio ingestion

- Extract: transcription into segments (start_ms, end_ms, text)
- Chunk: align chunk boundaries to segments when possible
- Pointer: pointer_type=AUDIO_MS, start/end in ms
- Time:
- if recording start time known: chunk_time = recording_start + offset
- else: use source_time fallback rules

Document ingestion (PDF/MD)

- PDF: extract per page; pointer PDF_PAGE (p_start, p_end)
- MD: split by headings; pointer can be (heading_id, offsets) or note-range offsets
- Store doc metadata (author/created if available) in metadata

Web ingestion

- Fetch URL with safety controls (timeouts, content-type allowlist)
- Extract main content; pointer URL (url + section/snippet id)
- Metadata: title/domain/published time (when available) + crawl time

Plain text

- Store as NOTE; pointer NOTE_RANGE offsets
- Time is usually strong: note creation time or user-supplied source_time

Images (required proposal: store + make searchable)

- Store image in object storage
- Generate associated searchable text:
- OCR text (screenshots/scans)
- Caption (vision model summary)
- User tags
- Index this text into chunks with pointer_type=IMAGE_REF
- Result: searchable by description (“that screenshot with the k8s error”).

Information retrieval & querying strategy (core)

Chosen strategy: Hybrid retrieval + temporal filtering (justified)

For query q:

1. Parse time constraints (if present)
2. Retrieve candidates using:
- Semantic: vector similarity on embeddings
- Keyword: Postgres FTS ranking
3. Apply time window filter/boost
4. Fuse rankings and select diverse evidence

5. Use LLM to synthesize answer from evidence and return citations

Justification

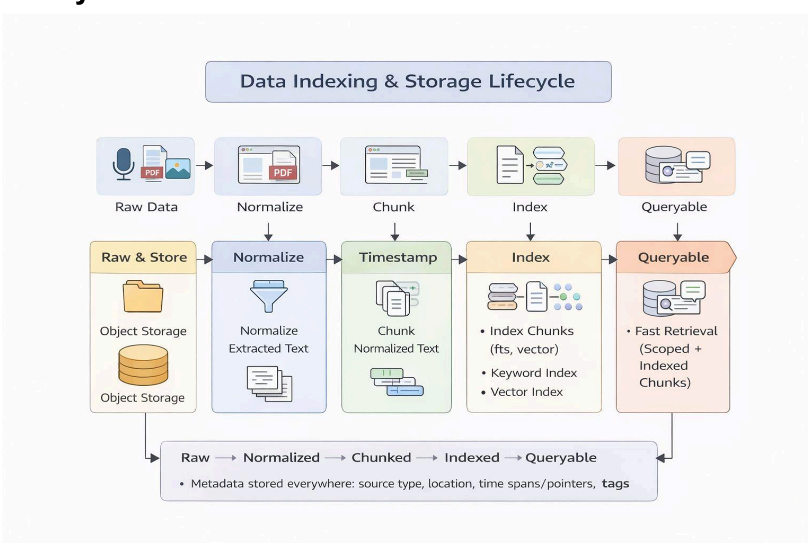
- Vector search improves recall on paraphrases and “meaning-based” queries
- Keyword search improves precision on exact terms and names
- Time filtering is essential for temporal correctness
- Single datastore keeps the system simple and coherent

Evidence selection rules (prevents “chunk soup”)

- Cap chunks per item_id to avoid dominance by one document
- Prefer chunks within the requested time window
- Always include pointers for citations

Data indexing & storage model (lifecycle + trade-offs)

Lifecycle



What metadata is stored

- Source type and source URI (filename / URL)
- Title/domain/author/tags (JSONB)
- Ingestion status and errors
- Timestamps (ingested, source/event time, chunk times)
- Citation pointers (page/time/url offsets)

Storage choice trade-offs

Postgres + pgvector + FTS (chosen)

- Pros: one operational surface; easy joins for user/time filters; strong schema; citations and metadata are first-class
- Cons: a dedicated vector DB can outperform at very large scale; can be introduced later without changing the ingestion contract

Temporal querying support

Time model

Store two concepts of time:

- Ingest time: ingested_at (always)
- Event/content time: source_time (item-level) and chunk_time_* (chunk-level)

This prevents confusion when older materials are ingested later.

Timestamp attribution precedence

For each item/chunk:

1. User-provided timestamp
2. Source metadata (recording time / doc metadata / web published time)
3. Extracted from content (explicit date strings; relative dates resolved using user timezone)
4. Fallback: ingested_at

Using time in retrieval

- Parse query (“last Tuesday”, “last month”) → compute [start,end]
- Filter/boost chunks using:
1. chunks.chunk_time_start/end when present
2. else fallback to knowledge_items.source_time
- Answer uses only evidence within time window unless user asks for broader context

Examples supported:

- “What did I work on last month?”
- “Key concerns raised in the project meeting last Tuesday?”

Scalability and privacy

Scaling to thousands of documents per user

- Async ingestion separates heavy processing from request path
- Per-user throttling in workers prevents noisy-neighbor issues
- Efficient indexing:
- FTS GIN index for keywords
- pgvector index for semantic retrieval
- time indexes for temporal filters
- Cost control:
- chunk size policy to manage index growth
- checksum/chunk_hash dedupe to avoid repeated compute

Privacy by design

- Every table row includes user_id; every query includes WHERE user_id = current_user
- Optional: Postgres Row-Level Security (RLS) for enforcement at the DB layer
- Raw artifacts stored in private object storage; avoid logging raw content
- Secure URL ingestion: prevent SSRF (block internal IP ranges), enforce timeouts and size limits

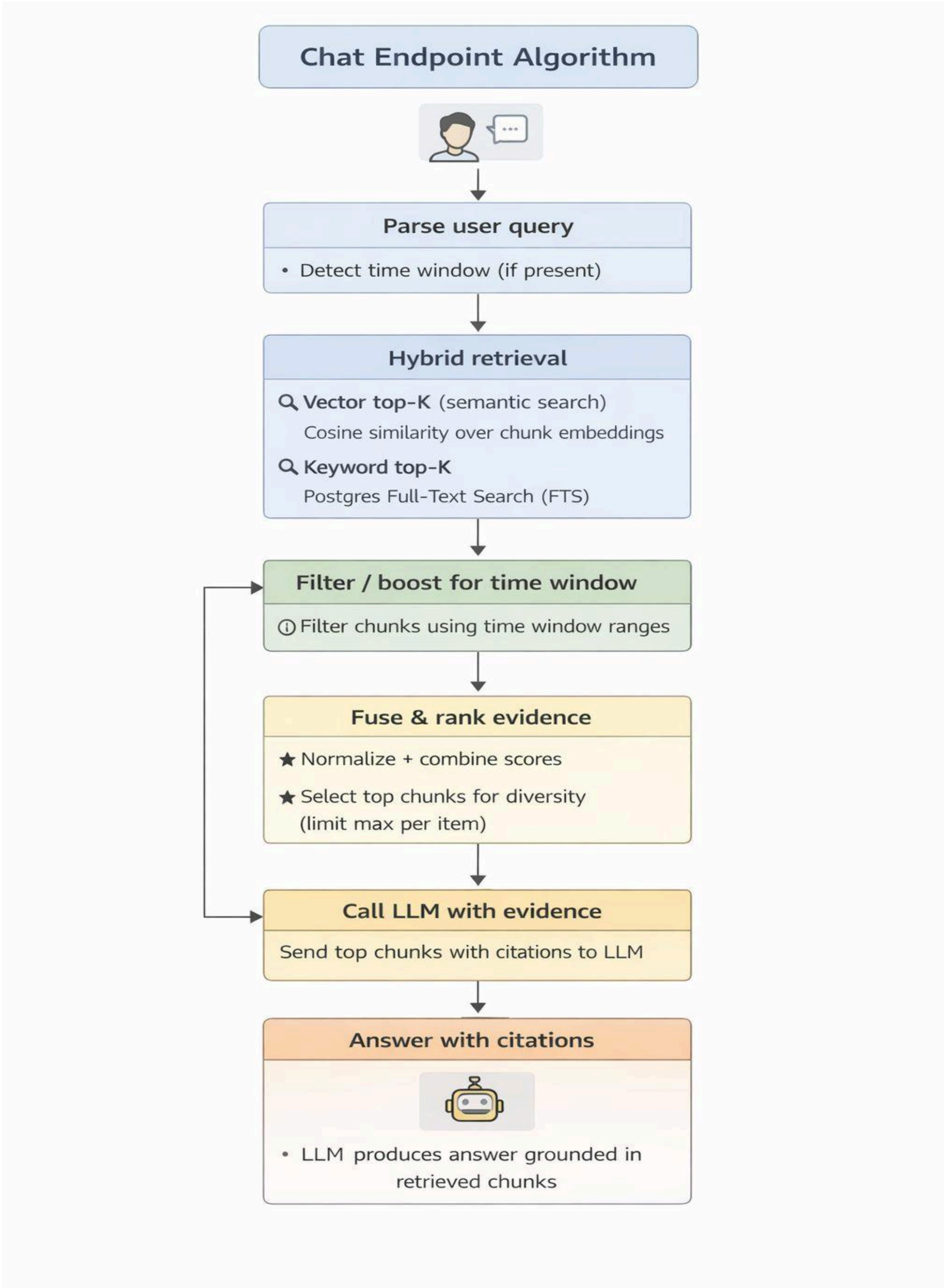
Cloud-hosted vs local-first trade-offs

- Cloud: best availability and compute; requires strong isolation/encryption
- Local-first: strongest privacy; can run storage/transcription/embeddings locally; same contracts remain, only components swap

- POST /chat/stream → SSE stream tokens + final citations

Q&A implementation

Chat endpoint algorithm


Chat Endpoint Algorithm
Parse user query
• Detect time window (if present)
Hybrid retrieval
Vector top-K (semantic search)
Cosine similarity over chunk embeddings
Keyword top-K
Postgres Full-Text Search (FTS)
Filter / boost for time window
Filter chunks using time window ranges
Fuse & rank evidence
Normalize + combine scores
Select top chunks for diversity
(limit max per item)
Call LLM with evidence
Send top chunks with citations to LLM
Answer with citations
• LLM produces answer grounded in
retrieved chunks