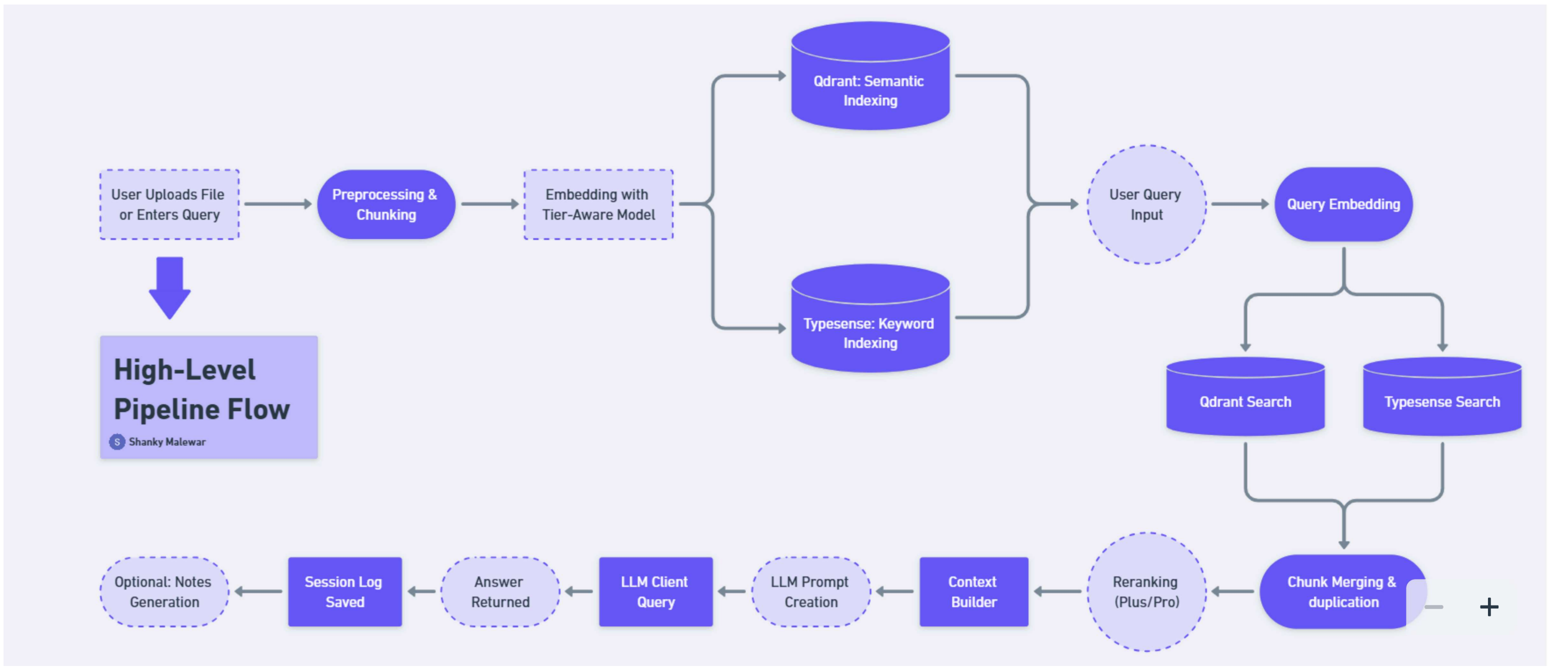
🧠 Asklyne RAG Architecture — Detailed System Design!

Asklyne's core AI capabilities are powered by a custom `Retrieval-Augmented Generation (RAG)` system that goes far beyond traditional static LLM chat. It allows Asklyne to process and reason over user-uploaded **`PDFs, images, code, and text documents`** — providing meaningful, context-aware answers, summaries, and learning notes.

This document outlines how Asklyne's end-to-end RAG pipeline functions across multiple user tiers (`free` , `plus` , `pro`) and input types (`text` , `code` , `notes`).



➡️ 1️⃣ Upload & Preprocessing (/upload-file)

When a user uploads a file, the following steps take place:

Mode	Processing Strategy
Notes(images)	OCR using `Tesseract` to extract visible text
Code	Parsed using custom `extract_code_from_*` logic
Text(.pdf .txt)	Extracted using `pdfplumber` (for PDFs) or UTF-8 decode

➡️ 2️⃣ Chunking (Token-Aware Splitting)

To ensure the LLM can work within token limits, raw content is split into semantically meaningful chunks using the `Chunker` class:

🧩 `Chunking Logic`

- **Text Mode**: Sentence-based splitting with overlap (max_tokens = 480, overlap = 80)

  - **Code Mode**: Function/class-based splitting using regex anchors on def and class
  - **Overlap** ensures contextual continuity between adjacent chunks for more coherent retrieval

---

➡️ 3️⃣ Embedding the Chunks

Chunks are transformed into high-dimensional vectors using tier-specific sentence encoders from sentence-transformers. This is handled by the Embedder class.

Tier	Text Embedding	Code Embedding
Free	`BAAI/bge-large-en-v1.5`	`microsoft/codebert-base`
Plus	`BAAI/bge-large-en-v1.5`	`Salesforce/codet5-base`
Pro	`intfloat/multilingual-e5-large`	`microsoft/graphcodebert-base`

➡️ 4️⃣ Storage in Dual-Engine Vector DBs

Asklyne indexes chunks in **two complementary retrieval systems**:

📦 Qdrant (Semantic Search)

- Stores vectors for **cosine similarity search**
- Collections per (tier, mode): e.g. `asklyne_chunks_plus_code`
- Embeddings are stored using `PointStruct` along with metadata (`session_id`, `mode`, `tier`, `text`)
- Supports filtering by session for scoped queries

🔎 Typesense (Keyword Search)

- Full-text index of each chunk for **keyword-based matching**
- Schema includes `text`, `tier`, `mode`, `session_id`
- Complements Qdrant by catching lexical matches not captured by vector similarity

This **hybrid retrieval strategy** ensures that both conceptual and literal matches are surfaced.

➡️ 5️⃣ Query Handling

When a user submits a query, Asklyne executes an orchestrated multi-step retrieval & generation process:

🧲 a. Retriever

- Embeds the query using the same encoder as the stored chunks
- Fetches top `k` relevant chunks from:
  - **Qdrant** (semantic match)
  - **Typesense** (keyword match)
- Deduplicates results by chunk text

📊 b. Reranker (Plus/Pro Only)

- For higher tiers, chunks are reranked using `cross-encoder/ms-marco-MiniLM-L-6-v2`
- Each chunk receives a `score` and is sorted accordingly

🧱 c. Context Builder

- Merges reranked top chunks into a prompt-safe block
- Uses 90% of tier's token limit to preserve headroom
- Format: `Chunk1\n---\nChunk2\n---\n...`

➡️ 6️⃣ Notes Generation (`/generate-notes` + `/generate-notes-pdf`)

Asklyne enables users to turn entire chat sessions into **clean, structured notes**:

✨ Features

- Generates notes using LLM prompting
- Markdown → HTML → PDF conversion
- Templated prompt includes:
  - Headings
  - Bullets
  - Concept highlights

📝 Modes

- `full`: Includes both Q & A
- `response_only`: Includes only answers
- `custom`: User-defined focus (e.g., "only summarize Python code explanations")

🔐 Tier-Aware Intelligence Routing

Every layer — chunker, embedder, reranker, LLM, note model — is **dynamically routed based on user tier**, enabling:

- Lower costs for casual learners
- Powerful tools for researchers & devs
- Custom logic per mode (`text`, `code`, `notes`)

This **modular and scalable architecture** allows Asklyne to balance cost and quality across diverse users.
