**High-level PFU RAG pipeline**

* Context sources to index
* All components catalog: from the existing types endpoint (preferred) or by scraping src/backend/base/langflow/components/\*\*.
* Core schema snippets: node/edge shapes and Graph.from\_payload expectations.
* Starter flows: a few curated .json examples for patterns.
* Indexing
* Build a “compact catalog” document per component with: name, type, base\_classes, minimal template (fields + \_type), key IO.
* Store in a lightweight vector store (in-memory FAISS or SQLite+pgvector). Expose a refresh endpoint.
* Keep 1–2 small static docs: node/edge minimal schemas; PFU ops contract.
* Request flow (per user prompt)

1) Summarize current flow to a “flow signature” (ids, types, small template subset).2) Retrieve K relevant components/templates + core schema snippets.3) Compose the larger prompt: kernel + flow signature + retrieved snippets + user prompt.4) Call GPT-5; expect PFU ops JSON.5) Validate ops; apply via POST /flows/{id}/ops with base\_version.6) Log/stream progress; console.log the model response for now.

* Minimal prompt kernel (example to embed)
* PFU role/goals, constraints (ops only), output format.
* Remind: don’t output full flow JSON; use provided templates to fill fields.
* Retrieval details
* Query = user prompt + flow signature + “needed capabilities” extracted (e.g., “LLM”, “retriever”, “router”).
* Retrieve top-K components (e.g., K=5–10) + their minimal templates; add core node/edge snippet once.
* Applying PFU
* Batch small ops (3–10 ops per round), compile check; on 400 return the error back to model for self-fix.
* Use updated\_at for optimistic concurrency.

**Where to put it**

* src/backend/base/langflow/airelius/
* router.py: endpoints
* POST /airelius/pfu/plan (exists): accept prompt, flow\_id, optional files; inject kernel; return composed prompt and kick plan loop later.
* POST /airelius/pfu/run: run full pipeline (compose → retrieve → LLM → apply ops).
* POST /airelius/pfu/index/refresh: rebuild index from types API.
* service.py: orchestration (compose\_prompt, retrieve\_context, call\_llm, apply\_ops).
* retriever.py: vector store wrapper (embed, upsert, top\_k).
* kernel.py: prompt templates and small schemas (PFU ops contract, node/edge minimal examples).
* summarize.py: flow → flow signature function.

**How to send files to the LLM**

* Don’t attach entire source files.
* Embed/index everything; retrieve only relevant component templates and a tiny schema doc.
* For “files from backend” in your prompt: send short, targeted snippets (component templates and minimal schema), not full files.

**Console logging**

* For now, POST /airelius/pfu/run returns:
* composed\_prompt
* retrieved\_snippets (ids/titles)
* raw LLM ops JSON
* applied result or validation error
* Also log to server console for visibility.
* Use RAG with a compact catalog: index all components; retrieve only relevant ones per request.
* Add airelius modules: service.py, retriever.py, kernel.py, summarize.py, extend router with /run and /index/refresh.
* Pipeline: flow signature → retrieve K templates + schema → compose kernel → GPT-5 → PFU ops → validate/apply → log.