FinQuery, a Grounded Personal-Finance Q&A Agent

Design Doc · Capstone, AI Engineering Cohort: RAG & Agents · KrantiKumar Gajula · July 2026

1. Problem Statement

The target user is anyone trying to understand **their own spending patterns** or learn **core personal-finance principles**. Getting this from a general-purpose chatbot today requires uploading raw bank statements, exposing names, account numbers, phone numbers, and UPI IDs to a third party. Answers computed over hundreds of transaction rows without code execution are also unreliable and can hallucinate.

Build: a grounded Q&A agent that answers two narrow classes of questions, while **guaranteeing the LLM never sees raw PII** (reversible pseudonymization layer, measured 0%-leakage target):

1. **Transaction Q&A**: "How much did I spend on food in June?", "Am I over budget on transport?" Answered with **exact figures computed from the user's transaction database**, with provenance ("₹18,240 across 47 transactions").
2. **Personal-finance education Q&A**: "What is the 50/30/20 rule?", "How does 80C deduction work?" Answered from a **curated, India-focused knowledge base**: public RBI investor-awareness booklets, SEBI/NCFE investor-education material, plus short original articles on budgeting principles.

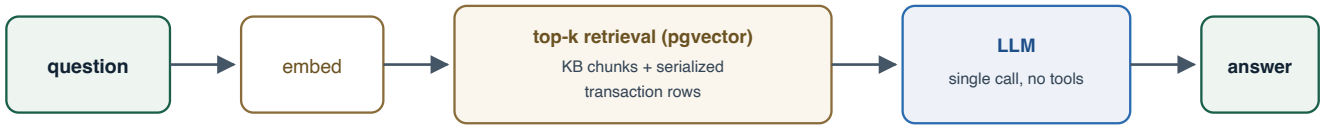
Explicitly out of scope: specific investment advice ("should I buy X?"), which is regulated in India (SEBI RIA); the system must detect and refuse. The refusal boundary is **specificity, not topic**: general principles applied to the user's numbers are education; product-level recommendations are refused. Also out: multi-turn memory, portfolio analysis, forecasting, and transaction categorization (data arrives pre-categorized).

Data is synthetic: a seeded Postgres instance, one user, ~6 months of realistic Indian transactions, reproducible with one `docker compose up`. Every transaction carries a category from a **fixed taxonomy** (Food & Dining, Transport, EMI, Rent, Utilities, Salary, ...) assigned at seed time; the taxonomy is included in the agent's system prompt so user phrasing ("eating out") maps to valid tool arguments.

2. Architecture

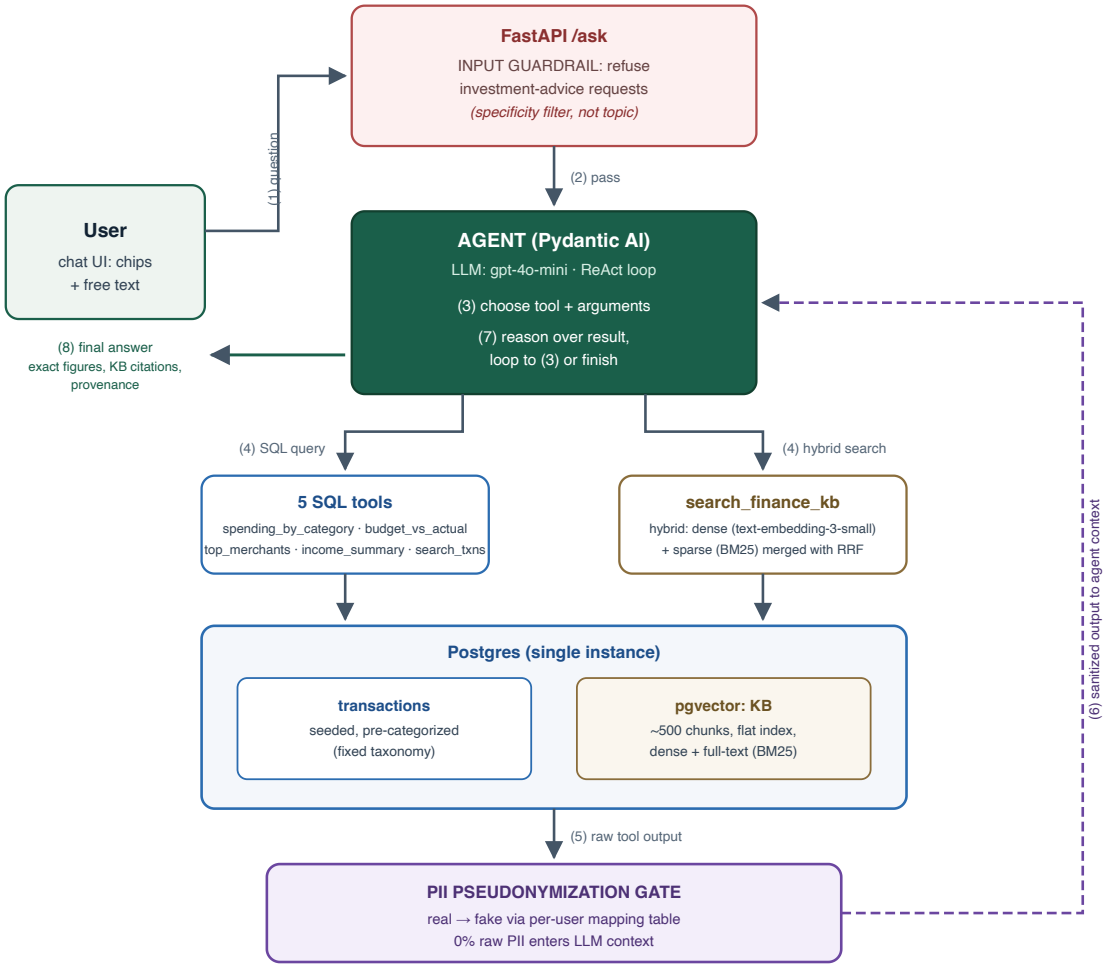
Two approaches, built and compared (mandatory comparative evaluation):

> **Approach A, Vanilla RAG (baseline).** Everything is a document: KB articles and serialized transaction records go through ingest → chunk → embed → index (pgvector). *Hypothesis: works for education questions, fails on aggregations; retrieval returns k transactions but the answer needs all N summed.*



Approach A: one retrieval pass, one LLM call; no tools, no loop.

> **Approach B, single tool-calling agent (ReAct loop).** One agent, six typed tools. Numbers are computed by SQL, never generated by the LLM. Cross-class questions ("After paying my EMIs I cannot save enough; how do I improve?") emerge naturally from the loop; vanilla RAG structurally cannot answer them.



Approach B: numbered sequence (1)–(8). Steps (3)–(7) repeat until the agent finishes; every tool output passes the PII gate before re-entering LLM context.

PII pseudonymization layer (core, applied to both approaches). Free-text transaction narrations (UPI strings) carry counterparty names, account numbers, phones, and VPAs. Before any tool output enters LLM context, a forward pseudonymization step replaces detected PII with consistent fake values from a per-user mapping table (detection: regex for account numbers, phones, and UPI VPAs, plus a known-values list; free-text name NER is documented as out of scope). Prior art is acknowledged (Microsoft Presidio). Reverse mapping (fake to real in final answers) is an **optional extension**: in-scope answers return numbers or principles, so pseudonyms rarely surface; it becomes required only if counterparty-name questions ("whom did I pay most?") are added.

Retrieval design (KB path). Hybrid search: dense (`text-embedding-3-small`) plus sparse (Postgres full-text/BM25), merged with **Reciprocal Rank Fusion**; evaluated as an ablation against dense-only. Chunking: ~400-token chunks, 15% overlap (short, self-contained articles). Index: exact/flat search; the corpus is ~500 chunks, so ANN indexes (HNSW/IVF) solve a scale problem this project does not have.

Considered and rejected: Graph-RAG/KAG (no entity-graph structure in the data; the structured side already lives in a relational DB), multi-agent (no coordination problem), map-reduce summarization and memory (no long docs, single-turn), re-ranking and HyDE (deferred; added only if evals expose retrieval misses they would fix, avoiding complexity before a baseline).

Stack: FastAPI service, Postgres + pgvector (one datastore for rows, vectors, and sparse index), OpenAI gpt-4o-mini, minimal chat UI (suggested-question chips plus free text, single-turn).

3. Evaluation Criteria

Golden set (~55 questions, 5 buckets), ground truth precomputed via SQL against the seeded DB: aggregation (15), lookup/filter (10), education (15), refusal probes (10), composite (5, requiring both the SQL and KB surfaces in one answer).

Metric	Type	What it measures
Numeric exact-match	Quantitative	Answer contains the exact figure from SQL ground truth (aggregation/lookup buckets)
Tool-selection accuracy	Quantitative	Correct tool and correct arguments, including valid taxonomy categories (agent only)
Tool-sequence correctness	Quantitative	Composite bucket: both surfaces (SQL and KB) were called
Retrieval hit rate	Quantitative	Gold chunk in top-k (education bucket; dense-only vs hybrid+RRF)
Refusal / false-refusal rate	Quantitative	Refuses advice probes; does not over-refuse legitimate questions
PII leakage rate	Quantitative	Every LLM-bound payload scanned against the real-PII mapping table; target **0%** (deterministic check, no judge)
Faithfulness, provenance, completeness, synthesis	LLM-as-judge (gpt-4o)	Narration states only what tools/chunks returned; citations present; synthesis quality on the composite bucket (no single ground-truth number exists there)

Comparison matrix: {Vanilla RAG, Agent + dense, Agent + hybrid-RRF} × 55 questions, all metrics reported per bucket; the aggregation and composite buckets are where the baseline is expected to fail hardest. Success thresholds: agent ≥ 95% numeric exact-match on aggregations, ≥ 90% refusal on probes, 0% PII leakage. Failures at each stage are documented in the README (Failure Analysis).

4. Framework Justification

- **Pydantic AI** (agent framework). The backend skill set is FastAPI + Pydantic; tools become typed, validated schemas with zero new idioms, and the agent loop is thin and inspectable, so orchestration decisions stay visible and gradeable.
- **pgvector over a dedicated vector DB**: one datastore serves relational queries, dense vectors, and sparse (full-text) search, making hybrid+RRF a SQL query instead of a second system; right-sized for a ~500-chunk corpus.
- **OpenAI gpt-4o-mini + text-embedding-3-small**: provided cohort API key; mini-tier is sufficient for tool-calling and keeps the ~165-run eval matrix affordable. LLM-as-judge uses gpt-4o for judge/actor separation.

Rejected, LangGraph: graph abstractions are justified by multi-agent or branching workflows, which this project explicitly rejected; here they would only hide the ReAct loop being evaluated.
Rejected, smolagents: a code-agent (the LLM writes and executes Python) is the wrong risk profile for a finance surface, and its main strength duplicates what typed SQL tools already provide deterministically and auditably.

Timeline: baseline + seed data (days 1-2), agent + tools + PII layer (days 3-5), hybrid ablation + eval harness (days 6-8), UI + demo video + docs (days 9-10).