Goal: Dynamic, Production-Grade Qdrant-Powered Vedabase

Final Features You'll Unlock:

- Semantic + hybrid search with filters like @type:artha, @book:rigveda
- Real-time querying via API by MCP/FinBot/Gurukul agents
- Persistent vector store on NAS with metadata
- Versionable and extendable KB (track updates, feedback loops)
- Future-ready for RLHF, user scoring, and topic clustering

Step-by-Step Plan (Today)

Karan

qdrant_loader.py + Retriever Wrapper

Role: Convert Vedabase PDFs → vector chunks → push to Qdrant

Tasks:

1. Mount NAS and read 2 Vedic PDFs (Rigveda, Atharvaveda)
2. Use pdfplumber or PyMuPDF to extract clean text
3. Chunk using LangChain RecursiveCharacterTextSplitter (400 tokens)
4. Embed using Instructor-XL (or fallback to OpenAI Ada)
5. Add metadata: {"book": "rigveda", "type": "artha", "version": "v1"}
6. Push vectors to Qdrant collection vedabase-v1
7. Save qdrant_loader.py for reusability
8. Export vedabase_retriever.py → expose get_relevant_docs(query, filters)

Nisarg

KnowledgeAgent with Qdrant

Role: Agent interface that takes in natural/structured queries → returns semantic + metadata-filtered chunks

Tasks:

1. Build KnowledgeAgent.py with __init__, .query() methods

2. Accept input: query_text, filters (e.g., {"type": "artha"})
3. Call vedabase_retriever.get_relevant_docs()
4. Return: text, score, source, timestamp, metadata
5. Log errors + fallback if no result

Nipun

POST /query-kb

API Endpoint + Logging

Role: Create the route inside MCP, wire to KnowledgeAgent, log queries to Mongo for analytics

Tasks:

1. Add /query-kb FastAPI route
2. Accept POST JSON: { "query": "...", "filters": {...} }
3. Call KnowledgeAgent.query(...)
4. Return JSON with text, metadata, score, trace_id
5. Log every query + response to MongoDB (kb_logs)
6. Include feedback: null field in logs for later RL

Vijay

File Access Module + Secure NAS Mount Check

Role: Ensure that MCP server has safe + permissioned access to read/write to NAS for agents

Tasks:

1. Confirm NAS is mounted at /mnt/vedabase/ or equivalent
2. Write helper function: read_pdf(path) → returns cleaned text
3. Add permission guardrails (e.g., read-only mode for agents)
4. Export this as file_utils.py in MCP utils
5. Pass file context to KnowledgeAgent if fallback needed

Vedant

Gurukul Integration Plan (Qdrant API Bridge)

Role: Create a lightweight call from Gurukul frontend/backend to the /query-kb endpoint for all KB calls.

Tasks:

1. Draft callKnowledgeBase(query, filters) in Gurukul backend (Node or Python)
2. Connect to /query-kb endpoint
3. Return structured response to UI component
4. Add loading + empty state handler in UI
5. Create a dummy “Ask the Vedas” button in UI to test

Additional Notes

- Qdrant Setup:
  - If not already running: docker run -p 6333:6333 qdrant/qdrant
  - Configure to store data on a mounted NAS volume for persistence
- Python Qdrant SDK:

```
pip install qdrant-client
```

- Embedding Hint:
  InstructorXL → sentence-transformers/instructor-xl
  Example:

```
model = SentenceTransformer("hkunlp/instructor-xl")
embeddings = model.encode([["Represent this document for
retrieval:", text]])
```