CASE STUDY #1

Assigning Specific Content to Specific Bots Without Letting Knowledge Bleed Between Them

Askme Bot -- Per-Bot Knowledge Scoping, Content Assignment and Query Isolation

Project	Askme Bot -- Customisable AI Chatbot Platform
Area	Multi-Bot Architecture, Knowledge Base Scoping, Query Isolation
Difficulty	High -- Multi-tenant content isolation, selective assignment, enforcement at query layer
Outcome	Strict per-bot content boundaries enforced at the query layer with no cross-bot knowledge bleed under any query pattern

Background

Askme Bot was designed around a simple but powerful idea: one account, multiple bots, each trained on exactly the content you choose. A business might run three bots from the same account: a customer-facing support bot trained on product documentation, an internal HR bot trained on company policies, and a sales assistant bot trained on pricing and competitor comparison sheets. Each bot served a completely different audience asking completely different questions.

The platform's value depended on users being able to trust that each bot answered strictly from its assigned content. A customer asking the support bot a question should never receive an answer that drew from the HR policy documents sitting in the same account. An employee using the HR bot should never see pricing information leak into a response because both sets of documents lived in the same underlying knowledge base infrastructure.

The Problem

Problem 1 -- Documents Lived in a Shared Pool With No Bot-Level Boundary

The first architecture stored all documents uploaded to an account in a single vector collection. When a user created a bot and marked certain documents as assigned to it, this assignment was recorded in a metadata table. At query time, a filter was applied to retrieve only chunks whose document ID appeared in the assignment list for the querying bot.

The problem was that this filter was applied after the vector similarity search, not before it. The search engine scanned the entire account's vector collection, found the most similar chunks across all documents, and then the application layer discarded chunks not in the assignment

list. The discarded chunks never reached the model, but they were retrieved and scored. This meant query performance scaled with total account document volume rather than assigned document volume, and more critically, it meant the assignment boundary was enforced by application code that could drift from the actual query execution path.

The Risk This Created

An application-layer filter is a soft boundary. It works until something goes wrong: a code change, a caching layer that serves a stale assignment list, or an edge case in the filter logic for newly assigned documents. In a platform where users trust that their HR bot will never surface customer pricing data, a soft boundary is not acceptable. The isolation needed to be enforced at the data layer, not the code layer.

Problem 2 -- Newly Uploaded Documents Were Temporarily Queryable by All Bots

When a user uploaded a new document, the ingestion pipeline chunked, embedded, and stored it immediately. The assignment step happened separately: the user would then navigate to a bot's configuration page and assign the document. In the window between ingestion completing and assignment being saved, the document's chunks existed in the shared collection with no assignment metadata. Any bot on the account that queried during this window could retrieve chunks from the unassigned document if they were semantically similar to the query.

For most users this window was seconds or minutes. For users who uploaded documents in bulk and assigned them later, or who forgot to complete the assignment step, it could persist indefinitely.

Problem 3 -- Removing a Document From a Bot Left Stale Chunks Retrievable During Cache Window

When a user removed a document assignment from a bot, the assignment record was deleted and the query filter was updated. However, the bot's query results were cached at the API layer for a short period to reduce repeated identical queries. A user who removed a document assignment and then immediately tested the bot might still receive answers drawn from the just-removed document because the cached response had not yet expired. There was no mechanism to invalidate the cache on assignment changes.

The Solution

1. Per-Bot Vector Namespaces for Hard Data Isolation

The shared vector collection was replaced with per-bot namespaces. When a user created a bot, a dedicated namespace was provisioned for it within the vector database. Documents were not stored in a shared pool and filtered by assignment. They were stored directly in the namespace of the bot they were assigned to. A query against a bot's namespace was physically scoped to that bot's data at the database level. No application-layer filter was involved. No other bot's documents existed in the namespace to be retrieved.

This meant isolation was enforced by the database, not by code. A bug in assignment logic, a caching anomaly, or an unexpected query path could not cause cross-bot data bleed because the other bot's documents were not present in the queried namespace to begin with.

2. Assignment-First Ingestion Flow

The ingestion pipeline was restructured so that embedding and storage only occurred after bot assignment was confirmed. When a user uploaded a document, the file was parsed and chunked immediately so the user could see its structure and preview its contents. The chunks were held in a staging area with no vector embeddings generated yet. The user then selected which bot to assign the document to. Only after the assignment was saved did the embedding and namespace storage run. There was no window during which an unassigned document's chunks existed in any bot's queryable namespace.

For users who uploaded documents in bulk and wanted to assign them later, the staging area stored the parsed chunks as pending. A dashboard indicator showed each user how many documents were staged but not yet assigned to any bot, prompting them to complete the assignment rather than leaving content in limbo.

3. Assignment Change Cache Invalidation

Any change to a bot's document assignments, whether adding or removing a document, triggered an immediate targeted cache invalidation for that bot's query cache. The invalidation ran synchronously before the assignment change was confirmed to the user, so by the time the user navigated to test the bot, the cache had been cleared and the next query would reflect the updated assignment. A bot that had just had a document removed would never serve a cached answer from that document because the cache was empty before the removal was acknowledged.

4. Assignment Audit Log Visible to Users

Each bot's configuration page included an assignment history log showing every document added or removed, who made the change, and when. This gave users a clear record of what each bot was trained on at any point in time, and gave teams accountability when multiple people had access to the same account. If a bot gave an unexpected answer, the first diagnostic step was checking the assignment log to see whether a document had been added or removed recently that might explain the change in behaviour.

Problem	Solution
Shared vector pool with app-layer filter is a soft boundary	Per-bot namespaces: isolation enforced at database level
Unassigned docs queryable during ingestion window	Assignment-first flow: embeddings only generated after assignment saved
Removed assignments still served during cache window	Synchronous cache invalidation on every assignment change

No visibility into what content each bot has access to	Assignment audit log with timestamp and user attribution per change

Outcome

After the namespace-based isolation architecture shipped, cross-bot knowledge bleed became structurally impossible rather than just unlikely. The per-bot namespace meant each bot's query was a closed operation over its own data. Users managing multiple bots for different audiences gained genuine confidence that their bots were independent. The assignment-first ingestion flow eliminated the unassigned document window entirely. The audit log gave teams managing shared accounts a clear view of exactly what each bot knew and when that had changed.

Key Takeaway

Content isolation in a multi-bot platform is not a feature to be enforced by application code. Code can be bypassed, can drift, and can fail in unexpected ways. The only isolation that holds reliably under all conditions is isolation enforced by the data storage layer itself. Per-namespace storage means a cross-bot query is not a policy violation waiting to be prevented. It is a physical impossibility. That is the right level of guarantee for a platform where users trust their bots to stay in their lanes.

-- Ehsan