You have a few solid architectural options for integrating your vector search feature with your Tauri frontend + LLM tool backend. The best choice depends on factors like deployment, modularity, development workflow, and performance expectations.

Let me break down the trade-offs of each option:

---

1. Vector Search as an API (e.g., Actix-web)

Pros:

- **Separation of concerns:** Your vector search service runs independently and exposes a clean HTTP/REST or gRPC API.
- **Language & Tech freedom:** You can build and deploy vector search in any stack without coupling it to Tauri.
- **Scalability & Reusability:** Other services or clients (not just Tauri) can call it easily.
- **Easier to scale/deploy separately:** You can scale vector search independently or deploy it on a dedicated server or cloud.
- **Better fault isolation:** Issues in vector search won't directly crash Tauri app.

Cons:

- Adds complexity in deployment (need to run and manage another service).
- Latency depends on IPC/HTTP calls (usually minimal locally, but must be considered).

---

2. Include Vector Search Code Directly in the Tauri Backend

Pros:

- Simple architecture, everything bundled together.
- Easier local integration with direct function calls — no IPC overhead.
- Single deployable binary for Tauri backend + vector search.

Cons:

- Tightly couples vector search logic to your Tauri backend, harder to maintain or reuse.
- Can lead to a monolithic codebase.
- Vector search might have heavier dependencies that bloat your Tauri backend binary.
- Less flexible scaling or separate upgrades.

---

3. Create a Workspace with Both Tauri and Vector Search as Separate Crates/Packages

Pros:

- Good middle ground — modular code but under one repo/workspace.
- You can share code (e.g., types, utils) between Tauri backend and vector search.
- Easier local development and testing — start/stop vector search as a separate process.
- Can still deploy vector search as a separate service (or embed it) depending on config.
- Cleaner project organization, better maintainability.

Cons:

- Slightly more complex than a single repo but manageable.
- Still need to decide on communication between Tauri backend and vector search (API calls, message passing, or direct calls if in the same binary).

Recommendation

If your vector search is substantial, possibly resource-intensive, or might be reused by other services/clients, I'd recommend going with option 1 (API/Actix-web). It's cleaner in the long term and better for scalability.

If you want simpler local integration and vector search is lightweight, option 3 (workspace with separate crates) gives you flexibility and maintainability, without fully committing to a separate service.

Avoid fully embedding vector search directly inside the Tauri backend (option 2) unless it's very small and tightly coupled, because it hurts modularity and maintainability.

Extra thoughts:

- You could start with a workspace approach (option 3), build your vector search as a library or small binary, and expose a local HTTP server or an IPC mechanism.
- Then your Tauri backend calls it via HTTP or native Rust API.
- Later, you can separate it as a standalone API service if needed without major rewrites.

- For local apps, if vector search is heavy, running it as a background local service (API) is common.
