Orion — Implementation & Code Tradeoffs

Orion Project — Implementation-level Discussions & Tradeoffs Participants: Dev X, Dev Y, Dev Z Dev X: Dense embedding job (dense_embeddings.py) reads PDFs and writes to Chroma — but I noticed error handling is minimal. If a PDF or a network call fails mid-run, partial state may be written. We should introduce transactional updates: build the new index in a temp workspace and only swap on success. It's slightly more work but avoids half-broken DB state. Dev Y: Good point. Also, there are multiple scripts that duplicate document-loading logic (aggregate_documents.py, get_relevant_docs.py). That invites drift. I'd consolidate document I/O behind a single DocumentLoader class with consistent metadata keys (source, time, page). That reduces bugs in sorting/time parsing we saw before. Dev Z: Regarding answer.py and rank_documents.py — the repo currently uses a reranking step. We need to be explicit about evaluation metrics. Is reranking improving MRR or just adding latency? For small datasets it's fine, but at scale the extra LLM calls are expensive. Consider making reranking optional with config flags that record latency/accuracy tradeoffs in benchmarks. Dev X: There's also a security angle: local scripts include a supabase_client.py with connection strings potentially in env files. We must ensure no secrets are committed and provide a .env.example in README. Also credentials in logs need scrubbing. Dev Y: For oterm (terminal capturing), it's neat but fragile — relying on `script` and requiring `exit` to flush data is error-prone. We should add explicit flush/rotate logic and unit/integration tests simulating sudden termination (SIGINT/SIGTERM). Tradeoff: more complexity vs reliability of captured command history. Dev Z: On testing and CI: the repo lacks unit tests around core behavior. Start with document loader, embedding interface (mock embedder), and simple query-response smoke tests. Add a small fixture dataset and run in CI with a tiny, deterministic embedding stub for reliability. This makes it easy to test refactors (e.g., migrating from Chroma to pgvector). Dev X: Last thought — make data migrations first-class. When we change metadata schema, we should ship a migration script that can upgrade older Chroma or sparse pickle formats. Otherwise contributors will be blocked by incompatible artifacts.