

An All-Purpose, Backend-Neutral Embedding Layer for OMOP Vocabularies

Nico Loesch¹, Richard Boyce², Georgina Kennedy^{1,3}

¹ School of Clinical Medicine, University of New South Wales, Australia,

² University of Pittsburgh, Pittsburgh, PA, USA

³ Ingham Institute, Sydney Australia

Background

Semantic search over medical vocabularies is useful for tasks such as concept grounding pipelines, interactive curation interfaces and agent-facing vocabulary services. There is an opportunity to improve this practice across the community by supporting groups who want to integrate semantic retrieval as a component of their tooling, without re-creating a bespoke embedding layer from scratch.

Most vocabulary search tooling is multi-method. Exact, normalised, or near match search handle high-confidence lexical matches, relationship traversal integrates structural and hierarchical context, with embedding search to filling the gap where lexical matching fails.

For the design of a retrieval layer that other tools can share as infrastructure to serve vocabulary embeddings, several properties matter:

- 1) The dominant usage pattern is write-once, read-many. The index is ideally built once (either as a batch, or on an as-needs basis through incremental updates) and then queried across many different workflows and access patterns. That asymmetry makes the read interface the more critical consideration for both usability and performance, especially when considering simple and flexible infrastructure requirements. Ideally, it should not have a hard requirement for a CDM connection nor a running embedding model to deliver its core functionality, while offering extended functionality that takes advantage of these resources when they are available.
- 2) Vocabulary coverage changes over time, so ingestion should support backfilling new or updated concepts without a full rebuild.
- 3) Multiple embedding models can be in use simultaneously for specific purposes. Support for this requires explicit model registration with immutable version identifiers rather than a single embedding table with variable naming/versioning functionality.
- 4) Performance requirements vary significantly between a researcher's laptop and a production grounding service, which argues for backend flexibility under a consistent interface.

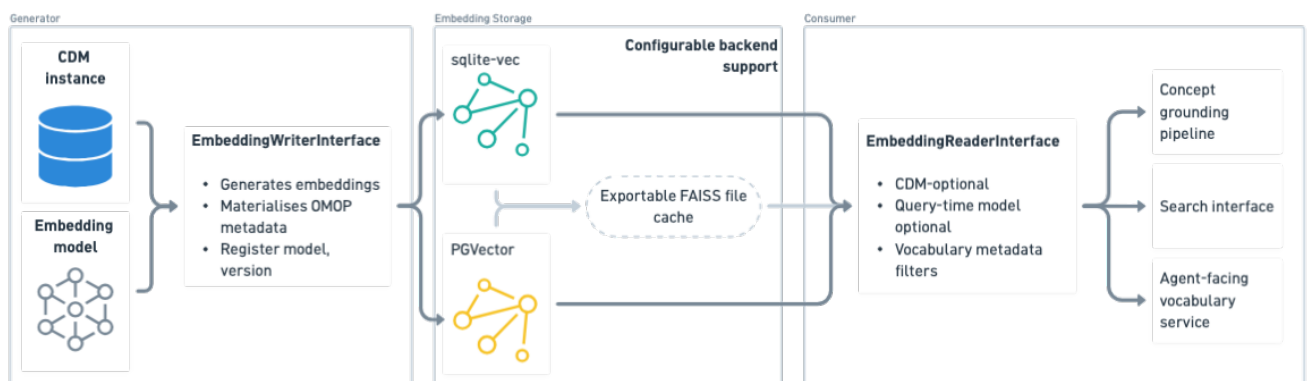


Figure 1: Generator / Storage / Consumer interface workflow of omop-emb

Methods

We developed omop-emb as an open-source Python library providing a reusable semantic retrieval layer scoped specifically to OMOP vocabulary concepts¹. The package is structured around two complementary interfaces for both generator and consumer roles (Figure 1), as well as an embedding storage backend.

EmbeddingWriterInterface (generators).

Generators use the writer interface to build and maintain an embedding index. During ingestion, OMOP metadata required for downstream filtering are materialised directly onto embedding records, so consumer-side query constraints can be applied as a pre-filter before the nearest-neighbour step, with no CDM round-trip required at query time.

Three storage backends are supported, each suited to a different deployment context (Figure 2). The default is `sqlite-vec`², which performs exact cosine similarity search and produces a single portable file with no external dependencies, making it appropriate for research workloads and corpora up to several hundred thousand concepts. Larger shared deployments are served by `pgvector`³ with HNSW (Hierarchical Navigable Small World) graph indexing. HNSW is an approximate nearest-neighbour structure that achieves sub-linear query complexity while maintaining high recall. Where running a live service is impractical, such as in air-gapped environments, HNSW-based FAISS⁴ indexes can be exported from either backend for distribution as self-contained artifacts that consumers can query without re-running ingestion. Trainable index types (IVF, IVFPQ), which reduce memory footprint through vector quantisation, are not yet supported.

EmbeddingReaderInterface (consumers).

Most groups will be consumers. The reader interface requires only a pointer to an existing backend. No CDM connection and no embedding model are required at query time. Both concept-to-concept nearest-neighbour retrieval and free-text-to-concept search are supported. Query-time filters are expressed in OMOP vocabulary terms and resolved within the embedding store.

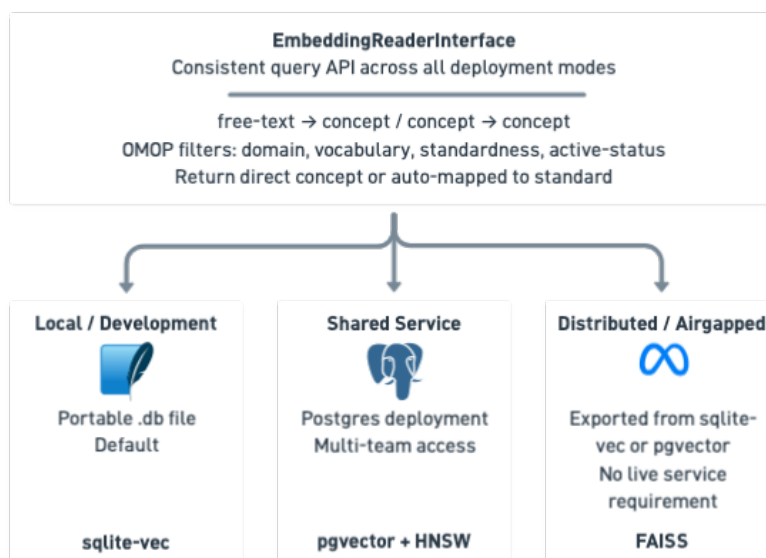


Figure 2: Use-case driven embedding backend selection

Results

`omop-emb`⁵ has been deployed as the semantic tier within a tiered concept grounding pipeline, where it operates as one channel alongside exact matching, full-text search, and cross-vocabulary mapping.

The resulting architecture is built around a stable retrieval interface across deployment modes. It can be deployed as a lightweight local setup with no separate service dependency, as well as being integrated within a richer search stack such as `omop-graph`⁶. The system supports both text-to-concept and concept-to-concept similarity search, together with in-database filtering by domain, vocabulary, standard status, and activity.

This gives a different outcome from embedding search implemented separately within each application. The same indexed concept space can support exploratory lookup, candidate generation

for concept grounding, and service-backed or agent-facing workflows without rebuilding the semantic layer for each task. Model choice and backend choice also become independent decisions: alternative embedding models can be evaluated side by side, and teams can move from local to larger-scale deployment without changing the retrieval contract exposed to downstream applications.

Accuracy Benchmark

We evaluated retrieval accuracy across six embedding models using 40 free-text queries representing lay terminology, historical eponyms, clinical abbreviations, and common misspellings. Evaluation was against a pool of approximately 10,000 standard *Condition* concepts, with success defined as retrieval of any acceptable standard concept ID within the top-5 results. Three indexing strategies were compared: embedding of concept names only (bare), embedding each synonym as a separate vector with max-pool aggregation at retrieval time (synonym_separate), and embedding a concatenated document of name and all synonyms (enriched). Results are shown in Table 1.

	bare			synonym_separate			enriched		
Model	MRR	R@1	R@5	MRR	R@1	R@5	MRR	R@1	R@5
text-embedding-3-large	0.805	0.775	0.85	0.8	0.75	0.85	0.786	0.75	0.85
mxbai-embed-large	0.451	0.375	0.55	0.631	0.6	0.7	0.531	0.5	0.575
medembed-large	0.467	0.4	0.55	0.615	0.575	0.675	0.425	0.375	0.5
bge-m3	0.232	0.2	0.3	0.6	0.575	0.625	0.556	0.525	0.625
MedEmbed-small	0.267	0.225	0.35	0.573	0.525	0.65	0.433	0.375	0.5
qwen3-embedding:0.6b	0.373	0.35	0.425	0.521	0.475	0.575	0.112	0.075	0.2
Hecate (API comparator)	0.696	0.65	0.75						

Table 1: MRR = mean reciprocal rank; R@1/R@5 = recall at rank 1/5. Evaluated against a ~10k-concept *Condition* pool, 40 queries.

For every local model tested, synonym_separate substantially outperformed bare, especially for smaller models. For the worst performing models, MRR improved by up to +0.37 MRR, which was a larger gain than switching to the strongest local model without changing index method. Second, a general-purpose proprietary model (text-embedding-3-large) outperformed all specialist medical models on this task. Note that Hecate scores are artificially reduced due to having many more concepts in the same embedding space. It uses the same model and method as the best performing combination and therefore provides a comparison to demonstrate scaling effects for discrimination under larger sets.

Backend latency benchmarking

We benchmarked query latency (excluding model inference) across four backend configurations using 200 pre-computed query vectors at k=5 and k=100, with and without concept domain/standard filters. Results are in Table 2. FAISS HNSW out-performs pgvector for raw lookup, but post-hoc database filters add latency due to a post-search round-trip to the database for concept filtering.

Backend	k=5				k=100			
	unfiltered		filtered		unfiltered		filtered	
	p50	p95	p50	p95	p50	p95	p50	p95
pgvector HNSW	7	13.8	7.9	10.3	8.2	29	9.4	16.4
FAISS HNSW	2.6	5	98.3	115.1	5.1	8.1	97.7	109.8
FAISS flat (exact; cached)	56.9	74.6						
sqlite-vec (exact)	212867	228073						

Table 2: All FAISS measurements with in-memory index caching; sqlite-vec stub measurement (n=40). Latency includes concept metadata enrichment step.

Conclusion

omop-emb provides a stable candidate-retrieval interface that multiple OMOP-native applications can share. As embeddings are used in more vocabulary search, grounding, and agentive workflows,

inconsistency in the retrieval layer becomes a systems problem rather than a minor implementation detail.

The benchmarking results reinforce both the value and the limits of the current design. At the 1.2-million-concept scale, pgvector HNSW provides latency suitable for interactive use under all realistic filter configurations. FAISS export offers meaningful value for air-gapped and distributed consumers, but its post-filter architecture means it should not be treated as a drop-in replacement for pgvector when concept domain and standard-status constraints are required at query time. sqlite-vec, while helpful for small concept sets, hits a hard performance ceiling at production scale that calls for either a move to pgvector or architectural changes to how the local file index operates.

By treating OMOP concept embeddings as reusable infrastructure rather than a bespoke feature of whichever application needed them first, `omop-emb` provides a practical path toward standardised semantic retrieval across heterogeneous tools, while preserving the local-first deployment patterns that remain important in many clinical research settings.

References

1. Reich C, Ostroplets A, Ryan P, Rijnbeek P, Schuemie M, Davydov A, Dymshyts D, Hripcsak G. OHDSI standardized vocabularies-a large-scale centralized reference ontology for international data harmonization. *J Am Med Inform Assoc.* 2024;31(3):583-90.
2. sqlitevz. sqlite-vec <https://github.com/asg017/sqlite-vec>
3. pgvector. pgvector <https://github.com/pgvector/pgvector>
4. Johnson J, Douze M, Jegou H. Billion-scale similarity search with GPUs. *IEEE Trans Big Data.* 2019;7(3):535-47.
5. Loesch, N., Kennedy, G., omop-emb: Vector embedding layer for OMOP CDM concepts. <https://github.com/AustralianCancerDataNetwork/omop-emb>
6. Loesch, N., Kennedy, G., omop-graph: a lightweight, opinionated knowledge-graph traversal and path-analysis library built on top of the OMOP vocabulary model <https://github.com/AustralianCancerDataNetwork/omop-emb>