

Tomefile: A Portable File Format for Prebuilt Local Retrieval Indexes

Simeon Emanuilov

Faculty of Mathematics and Informatics, Sofia University “St. Kliment Ohridski”, Sofia, Bulgaria
ssemanuilo@fmi.uni-sofia.bg ORCID: 0000-0002-2295-4513

Abstract

Organisations that want a local language model to answer from their own documents often try to fine-tune it on those documents, although prior work shows that fine-tuning is an unreliable way to add new facts and that retrieval is the more dependable route. Retrieval has a distribution problem of its own: the index lives inside one application or one database server, so every new reader chunks and embeds the corpus again, and must use the same embedding model for the vectors to mean anything. We present Tomefile, a file format and a reference command-line tool for handing over a finished retrieval index. A `.tome` file is a store-only ZIP archive that holds a SQLite database of chunk text and `sqlite-vec` vectors, the embedding model as GGUF weights or as a SHA-256 reference to them, checksums, an optional Ed25519 publisher signature, and a manifest with a stable package identifier, a revision number, and license grants. `tomefile serve` opens the file and exposes an OpenAI-compatible chat endpoint on the loopback interface that returns the cited source spans with every answer. On an Apple M1, building a 3,072-chunk corpus takes 63.7 s, while opening the resulting file and retrieving the first context takes 0.58 s, independent of corpus size. On 256 questions about synthetic facts that no model has seen, a 1.5B-parameter model answers 0.0% correctly closed book and 100.0% through the file. An independent 83-line reader that shares no code with the reference implementation returns the same top-3 chunks as the gateway for 256 of 256 queries.

1 Introduction

A recurring request in organisations that adopt local language models is to make the model “know” an internal handbook, a product manual, or a body of regulations. A common first answer is to fine-tune a small open model on the documents, because parameter-efficient methods such as LoRA and QLoRA make this cheap [6, 14]. The result is usually a model that has picked up the vocabulary and tone of the documents and still answers fact questions wrongly. This matches the literature: unsupervised fine-tuning injects new knowledge less reliably than retrieval [30], examples that carry new facts are learned slowly and increase hallucination once learned [12], and models struggle with facts that are rare in their training data [16, 21]. Retrieval-augmented generation (RAG) keeps the facts outside the weights, in an index that can be read,

cited, and replaced [10, 19].

Retrieval, however, is rarely shipped as an artifact. In local desktop tools the index lives in application data [23, 25, 28]. In server deployments it lives in a database such as PostgreSQL with `pgvector` [17] or `Qdrant` [33]. A colleague, a customer, or a second machine that needs the same knowledge receives the documents and rebuilds the index. That rebuild has a hidden coupling: stored vectors are comparable only with query vectors from the *same* embedding model, so the receiver must also obtain that exact model. A file of vectors without the identity of its embedder cannot be queried correctly.

This paper presents Tomefile, a format and a reference implementation built around one sentence: *build the index once, hand someone the file*. Its contributions are:

1. An archive layout, the `.tome` file, that carries chunk text, vectors, and the embedder identity by SHA-256, with a stable package identifier and revision (Section 4).
2. A binding rule and a short list of conformance requirements that let an independent program open the file and retrieve correctly (Section 4.1).
3. A consumption interface: the OpenAI chat-completions protocol [29] on `127.0.0.1`, with source spans attached to every answer, so existing clients work unchanged (Section 5).
4. An evaluation of the handoff cost, of answer accuracy with and without the file, and of an independent reader (Section 6).

Tomefile does not propose a new retriever, a new chunker, or a faster inference engine. Inference runs in `llama.cpp` [13] or `Ollama` [26], and retrieval quality is that of the embedder named in the file. Chunk text is stored in plaintext, so the format is not a way to protect a corpus.

2 Facts Belong in the Index

Questions such as “which code opens the main door of the Plovdiv office?” are lookups. Their answers must be exact, they change at the next revision of the document, and the person asking needs to see where the answer came from. Fine-tuning is well suited to style, output format, and narrow behaviours. It is poorly suited to these three properties. A fact absorbed into the weights cannot be cited, cannot be removed without retraining, and is recalled unreliably [1, 12, 31]. In a controlled comparison, retrieval outperformed unsupervised

fine-tuning both for knowledge the model had seen and for entirely new knowledge [30].

We do not repeat a fine-tuning comparison here. Instead, Section 6.3 measures the two quantities that matter for a hand-off: what a small local model knows about a private corpus without the file (closed book), and what the same model answers when the file supplies the context. The corpus is synthetic, so the model’s prior knowledge of its facts is zero by construction.

3 Related Work

Application-local RAG. AnythingLLM, Open WebUI, and GPT4All let a user add documents and chat with them on one machine [23, 25, 28]. The index is kept in the application’s own storage; it is not designed as an artifact that another program opens.

Local model runners. Ollama, LM Studio, and llamafile [24, 26] make a chat model easy to run. llamafile packages weights and server in one executable, which is the closest analogue for models; none of these define a knowledge index that travels with its embedder.

Portable indexes. Several recent projects share Tomefile’s goal. RagCap stores sources, chunks, and embeddings in one SQLite file and offers a CLI and an HTTP API [2]. RagBucket writes a .rag archive with a FAISS index [7], a JSON chunk list, and a manifest [4]. lance-bundle packages LanceDB vectors together with an ONNX export of the Sentence-Transformers model that produced them [5]. txtai saves an index and its configuration as a compressed archive that can be published on the Hugging Face Hub [22]. Tomefile differs in the combination it fixes: a pinned SQLite schema readable with the standard library plus `sqlite-vec` [9, 11], an embedder in the GGUF format that runs in `llama.cpp` without a Python machine-learning stack, a package identifier with a monotonic revision and license grants in the manifest, and the OpenAI protocol as the query interface. We see these projects as evidence that the problem is real, not as competitors to be displaced. A shared, independently readable format is more useful than any single tool.

Hosted vector stores. PostgreSQL with `pgvector`, Qdrant, and similar systems are how one operates a live, shared index, usually with approximate search structures such as HNSW [20]. They handle concurrent writers, access control, and an UPDATE when one page changes. Qdrant snapshots move a collection between servers without re-indexing [33], but the receiver still runs a Qdrant server, and the collection records the vector size and distance, not the model that produced the vectors. A .tome file is the opposite trade-off: a finished revision that the receiver opens locally with no server of the publisher’s, and whose embedder is pinned by hash. A corpus edited in place by many people belongs in a hosted store.

Package registries. OCI registries, KitOps ModelKits, and the Hugging Face Hub version and distribute model and dataset bytes [18, 27]. They are suitable places to store .tome files. They do not define a retrieval schema.

```
handbook.tome      store-only ZIP (method 0), ZIP64 if needed
|-- manifest.json  identity, embedder, settings, licenses
|-- checksums.sha256  SHA-256 of every other member
|-- signature.json  optional: Ed25519 over checksums.sha256
|-- models/
|   |-- embed.gguf   optional: omitted when referenced by hash
|   |-- chat.gguf    .tbox only, and only when embedded
|-- knowledge/
|   |-- vectors.db   SQLite: chunks, tome_meta, vec_chunks
```

Figure 1: Archive layout. Every member is stored uncompressed, so a reader can locate and hash members without decompression.

Table 1: Manifest fields in format version 1.3.

Field	Meaning
version, type	1.3; data_tome or tomebox
id, revision	Stable package name; integer ≥ 1
models.embed	Member path, or {sha256, filename}
models.chat	.tbox only; same two forms
embedding	Model id, sha256, dimensions, metric (cosine)
rag_settings	chunk_size, retrieve_top_k, system_prompt with {context} and {query}
provenance	builder, builder_version, chunker
license.*	Grants for knowledge, embed, chat: name, commercial, redistribute, attribution

4 The Tomefile Format

A .tome file holds knowledge and the identity of its embedder. A .tbox file uses the same layout and adds a chat model. The current format version is 1.3; the database schema version is 1. Figure 1 shows the layout.

Archive. The container is a ZIP archive [32] whose members all use compression method 0 (store). Writers use ZIP64 only when sizes require it; readers accept both. Names are UTF-8, relative, use /, and must not be absolute or have a .. path segment. The reference writer sorts members and sets every timestamp to 1980-01-01.

Manifest. manifest.json is UTF-8 JSON. Unknown fields are ignored and missing required fields are an error. Table 1 lists the fields. The package id is a reverse-DNS name that stays fixed across rebuilds; revision is an integer that the publisher increments. The SHA-256 of the archive bytes identifies one exact file and serves as the cache key, so it is not stored in the manifest. Subscribers follow id and revision; caches follow the archive hash.

Database. knowledge/vectors.db is one SQLite file written with `journal_mode=DELETE` and checkpointed, so the archive never contains -wal or -shm files. It has three tables:

```
CREATE TABLE tome_meta (key TEXT PRIMARY KEY, value TEXT NOT NULL);
CREATE TABLE chunks (id INTEGER PRIMARY KEY, text TEXT NOT NULL,
  source TEXT NOT NULL, start INTEGER NOT NULL,
  "end" INTEGER NOT NULL, sha256 TEXT NOT NULL);
CREATE VIRTUAL TABLE vec_chunks USING vec0(
  embedding float[384] distance_metric=cosine);
```

chunks.id equals vec_chunks.rowid; source is the document path relative to the build input; start and end are character offsets into that document. tome_meta repeats the package id, revision, dimensions, metric, embedding id, and embedding

hash, and records the sqlite-vec version used at build time. Retrieval is a single exact k -nearest-neighbour query:

```
SELECT c.id, c.text, c.source, c.start, c."end", v.distance
FROM vec_chunks AS v JOIN chunks AS c ON c.id = v.rowid
WHERE v.embedding MATCH ? AND k = ? ORDER BY v.distance;
```

Two further tables, `graph_nodes` and `graph_edges`, are optional and ship empty. Vector retrieval does not read them, and a file without them is valid.

Integrity and signature. `checksums.sha256` lists every member except itself and `signature.json`, in GNU sha256sum form and sorted by path. The optional `signature.json` holds an Ed25519 signature [15] over the exact bytes of `checksums.sha256`, the signer’s reverse-DNS name, and the public key. A signature that verifies proves that the file is unchanged since the holder of that key signed it. It proves who the publisher is only when the receiver already holds that public key from the publisher, obtained separately; the reference tool reads such keys from a local trust store and, with `--require-signature`, refuses unsigned files and files signed by any other key.

Models by reference. A model is either an archive member or a reference `{sha256, filename}` resolved from a content-addressed local cache. References let several files share one copy of the weights and let a publisher ship a `.tbox` without redistributing a chat model whose license forbids it. The reference packer refuses to embed a chat model unless the publisher marks it redistributable.

4.1 Conformance

The keywords below follow RFC 2119 [3]. A conformant reader:

- R1** MUST reject a member that is compressed, has an absolute name, or has a `..` path segment.
- R2** MUST verify every hash in `checksums.sha256`, and that the listed names equal the archive members other than `checksums.sha256` and `signature.json`, before it reads the database or loads a model.
- R3** MUST resolve the embedder (member or reference) and refuse to query unless its SHA-256 equals `embedding.sha256`.
- R4** MUST open `vectors.db` read-only and refuse it when `tome_meta` disagrees with the manifest.
- R5** MUST embed the raw query text with that embedder, with no model-specific prefix, and run the query above with $k = \text{retrieve_top_k}$.
- R6** SHOULD report source, start, and end of every chunk it uses.
- R7** MUST reject a present signature that does not verify, and MAY require a trusted signer.

R3 is the binding rule. Without it, a reader can return plausible but wrong neighbours from a model with the same dimensionality.

5 Reference Implementation

The reference implementation is a Python command-line tool, `tomefile`, built on Typer, FastAPI, httpx, the standard `sqlite3` module, and `sqlite-vec`. Models run in `llama-server` from `llama.cpp`.

Build. `tomefile build` reads `.txt` and `.md` files under a directory, in sorted path order, as UTF-8. The fixed-size chunker cuts each file into windows of 500 characters without overlap, ending a window at its last whitespace when the file continues. Chunk ids start at 1 and follow the same order. The tool starts the embedding GGUF in `llama-server --embeddings` on an ephemeral loopback port, embeds each raw chunk through `POST /v1/embeddings`, takes the dimension from the first vector, and writes the database and the archive. The embedder is stored in the archive unless it is already in the local model cache, in which case the manifest references it by hash; `--embed-model` forces the copy.

Serve. `tomefile serve` hashes the archive, verifies the checksums and any signature, extracts the manifest and the database into a cache directory named by the archive hash, and resolves each model in a content-addressed model cache, `~/.cache/tomefile/models/<sha256>.gguf`, copying an embedded model there once. It checks `tome_meta` against the manifest and the embedder hash against `embedding.sha256`, starts the embed worker and, for the internal engine, a chat worker, waits for both health checks, and only then binds the gateway to `127.0.0.1`. With `--engine ollama` the chat step goes to a running Ollama instance instead.

Request path. For `POST /v1/chat/completions`, the gateway takes the last user message as the query, embeds it, retrieves the top k chunks, joins their text with blank lines in distance order, and substitutes it for `{context}` in the manifest’s system prompt. It forwards the conversation’s user and assistant messages, drops client system messages, passes temperature and stream through, and ignores `model`. Every response, including each streamed chunk, carries a `tomefile` object with the package id, revision, the source, start, and end of each retrieved chunk, and the knowledge license name when attribution is required. `GET /v1/models` returns the package name, which is what OpenAI-compatible clients query before their first message.

Other commands. `tomefile pack` turns a `.tome` into a `.tbox` by adding a chat model reference. `tomefile inspect` verifies a file and prints its identity, models, counts, and member sizes, and with `--chunks` prints every stored chunk. `tomefile graph` loads optional graph rows from a JSON file; `serve --graph` then adds at most k chunks one edge away from the vector results. Graph retrieval is off by default and is not evaluated here.

6 Evaluation

The evaluation asks three questions. What does the receiver save by getting the file instead of the documents? Does the file let a small local model answer facts it cannot know? Can a program that does not share code with the reference implementation read the file and get the same results?

6.1 Setup

All runs used one Apple M1 with 8 GB of memory, macOS 26.6.2, Python 3.12.12, SQLite 3.51.3, `sqlite-vec` 0.1.9, `llama.cpp` build 6710, and `tomefile` 0.1.0. The embedder was BGE-small-en-v1.5 [35] in Q8_0 GGUF (36.7 MB, 384

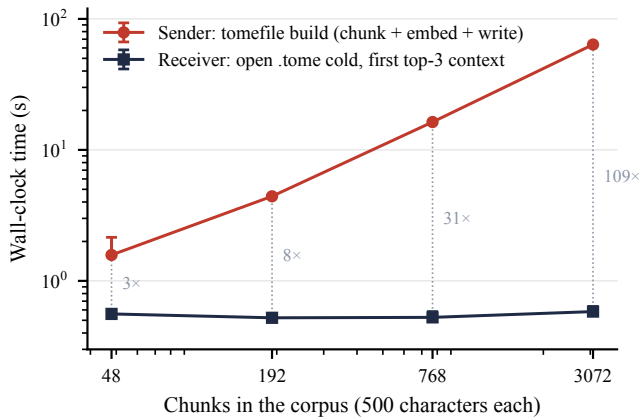


Figure 2: Sender build time and receiver cold-open time (median of 3; bars show min and max). Labels give the ratio. The receiver’s cost does not grow with the corpus.

dimensions, SHA-256 f046db1d. . .). The chat model was Qwen2.5-1.5B-Instruct [34] in Q4_K_M GGUF (1.12 GB, 6a1a2eb6. . .).

Corpus. We generated a fictional company handbook with a fixed seed. Each office has one Markdown file with four facts drawn at random and unique across the corpus: a four-digit door code, a guest wifi password of the form word-word-NN, an office manager’s name, and a four-digit IT extension. Every file also contains shared boilerplate, so the files are near-duplicates that differ mainly in the office name and the facts. Each file yields three chunks, and no fact crosses a chunk boundary. Because the values are random, no model can know them from pretraining. The question set covers every fact of a 64-office corpus (192 chunks, 256 questions). Questions are paraphrased rather than copied from the text: the file says “The guest wifi password at the Burgas office is . . .”, and the question asks “What should a guest type to join the wifi in the Burgas office?”.

Scoring. An answer is correct when it contains the gold value, compared case-insensitively. Retrieval succeeds when the chunk holding the gold value is among the chunks returned. All generation used temperature 0.

6.2 Handoff cost

For corpora of 16, 64, 256, and 1024 offices (48 to 3,072 chunks), we timed tomebuild on the sender and, on the receiver, the start-up steps of serve that a .tome needs before it can retrieve: verify and extract the file, start the embed worker, embed one question, and run the query. Starting a chat model costs the same with or without the file and is excluded. Each receiver run used an empty home directory, so caches started cold. Every measurement is the wall-clock time of a separate process, repeated three times. Transferring the file over a network is not included.

Figure 2 shows the result. Build time grows linearly at about 20.6 ms per chunk, from 1.58 s for 48 chunks to 63.7 s for 3,072 chunks. The receiver’s cold open stays between 0.51 and 0.64 s across all twelve runs. For the largest corpus, the process spent 89 ms importing modules, 164 ms hashing, ver-

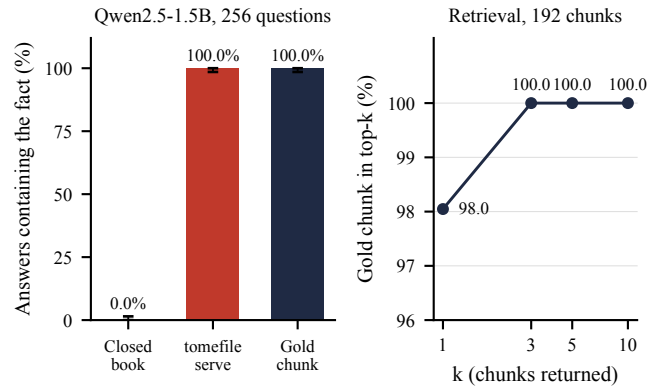


Figure 3: Left: answers that contain the gold value, with 95% Wilson intervals. Right: share of questions whose gold chunk is in the top k , measured with the independent reader.

ifying, and extracting the archive and copying the embedder into the cache, 242 ms starting the embed worker, and 22 ms embedding the question and searching (medians); the remainder is interpreter start-up. A warm open, with the embedder already cached, took 0.53 s. The archive grows from 38.3 to 43.2 MB; the embedded GGUF accounts for 36.7 MB of that, and the database grows from 1.66 to 6.46 MB for 1.15 MB of source text.

At 3,072 chunks the receiver saves a factor of 109 in time and, more importantly, does not need to know which embedder to install or how the corpus was chunked. The absolute saving is modest here because BGE-small is fast. It scales with corpus size and with the cost of the embedder, both of which grow in realistic deployments.

6.3 Answers with and without the file

We started tomebuild serve on the 64-office file with the Qwen model and sent each question three ways: (i) *closed book*, directly to the chat worker with no context; (ii) through the gateway, which retrieves the top 3 chunks; (iii) with the *gold chunk*, directly to the chat worker with the manifest’s prompt filled with the chunk that holds the answer. All three conditions used the same llama-server process. Conditions (i) and (iii) capped output at 128 tokens; the gateway passes no cap.

Figure 3 shows the result. Closed book, the model answered 0 of 256 questions correctly (0.0%, 95% CI 0.0–1.5%). Its answers were fluent: some invented a value (“The phone extension that reaches IT support for the Aberdeen office is 534”), others described how such codes usually work. Through the file it answered 256 of 256 (100.0%, CI 98.5–100.0%), the same as with the gold chunk (256 of 256). The gateway’s top 3 contained the gold chunk for every question, so no error remained to split between retrieval and reading. At depth 1 the gold chunk ranked first for 251 of 256 questions (98.0%, MRR@10 0.990); in the other five, all IT-extension questions, a different chunk of the same office ranked first and the gold chunk second. The median end-to-end latency through the gateway was 0.98 s per question, almost all of it generation; embedding a question and searching, measured separately with the reader of Section 6.4, took a median of 12 ms (95th

percentile 18 ms).

The gap between the first two bars is the argument of Section 2 in miniature: a small local model has no access to private facts, and the file supplies them without touching its weights. The equality of the second and third bars says that on this corpus the file’s retrieval costs nothing in accuracy. That ceiling is a property of the corpus, which is easy for retrieval because every question names its office. The experiment tests the handoff, not retrieval on hard queries.

6.4 An independent reader

A format becomes a standard when a second program can read it. We wrote a reader of 83 lines of Python that imports only the standard library, `httpx`, and `sqlite-vec`, and shares no code with `tomefile`. It implements R1–R5 (Figure 4) and opens both forms of the embedder: the timing corpora carry the GGUF as a member, and the evaluation file references it by hash, which the reader resolves from a local directory of models. The reader starts its own llama-server on the embedder it verified and retrieves for each of the 256 questions. Its top-3 chunk ids matched the chunks the gateway returned for 256 of 256 questions.

7 Limitations

The evaluation uses one machine, one embedder, one chat model, and a synthetic corpus whose questions name their target. It shows that the handoff works and what it costs. It does not measure retrieval quality on natural documents, where chunking and the embedder dominate. Substring scoring can credit an answer that contains the value in a wrong sentence; with random values this is unlikely but not excluded.

The format itself has deliberate limits. Chunk text is plain-text, so anyone holding the file can read the corpus. Version 1.3 ingests only `.txt` and `.md`. The receiver must run the exact embedder named in the file. This is the point of the format, but it couples a file to a model, and a new embedder means a new revision. Retrieval is an exact scan, which is fast at the sizes measured here but linear in the number of chunks. A file is a published revision: corpora that many people edit in place belong in a hosted vector store. A signature proves integrity since signing, and authorship only relative to a key the receiver already trusts. Finally, one implementation and one independent reader do not make a standard; that requires other tools to read and write the same bytes.

8 Conclusion

Tomefile treats a retrieval index as a document to be handed over, not as state inside an application. A `.tome` file pins its embedder by hash, carries a stable identity and revision, and opens in about half a second into an OpenAI-compatible endpoint that cites its sources. Facts stay in an index that can be read, cited, and replaced, and the person who receives the file does not rebuild it. The specification, the reference implementation, and the scripts behind every number in this paper are available at <https://github.com/s-emanuilov/tomefile> [8]. The project site is <https://tomefile.com>.

References

- [1] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.
- [2] Caleb Andersen and Archeta AI. RagCap: Portable RAG capsules. <https://github.com/ArchetaAI/ragcap>, 2025.
- [3] Scott Bradner. Key words for use in RFCs to indicate requirement levels. RFC 2119, Internet Engineering Task Force, 1997.
- [4] Anik Chand. RagBucket: Portable executable RAG artifacts for Python. <https://github.com/anikchand461/ragbucket>, 2026.
- [5] cloudkj. lance-bundle: Portable embeddings. <https://github.com/cloudkj/lance-bundle>, 2025.
- [6] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, 2023.
- [7] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The Faiss library. *arXiv preprint arXiv:2401.08281*, 2024.
- [8] Simeon Emanuilov. Tomefile: Specification and reference implementation. <https://github.com/s-emanuilov/tomefile>, 2026.
- [9] Kevin P. Gaffney, Martin Prammer, Larry Brasfield, D. Richard Hipp, Dan Kennedy, and Jignesh M. Patel. SQLite: Past, present, and future. *Proceedings of the VLDB Endowment*, 15(12):3535–3547, 2022.
- [10] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.
- [11] Alex Garcia. sqlite-vec: A vector search SQLite extension. <https://github.com/asg017/sqlite-vec>, 2024.
- [12] Zorik Gekelman, Gal Yona, Roei Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. Does fine-tuning LLMs on new knowledge encourage hallucinations? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7765–7784. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.444.
- [13] Georgi Gerganov and llama.cpp contributors. llama.cpp: LLM inference in C/C++, including the GGUF model format. <https://github.com/ggml-org/llama.cpp>, 2023.
- [14] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [15] Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (EdDSA). RFC 8032, Internet Engineering Task Force, 2017.
- [16] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large language models struggle to learn long-tail knowledge. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 15696–15707, 2023.
- [17] Andrew Kane and pgvector contributors. pgvector: Open-source vector similarity search for Postgres. <https://github.com/pgvector/pgvector>.

```

def open_tome(path, workdir, models_dir):
    z = zipfile.ZipFile(path)
    for i in z.infolist():
        # R1
        assert i.compress_type == zipfile.ZIP_STORED and not i.filename.startswith("/")
        assert ".." not in i.filename.split("/")
    listed = dict(reversed(line.split(" ", 1))
                  for line in z.read("checksums.sha256").decode().splitlines())
    assert set(listed) == set(z.namelist()) - {"checksums.sha256", "signature.json"}
    for name, digest in listed.items():
        # R2
        assert hashlib.sha256(z.read(name)).hexdigest() == digest, name
    manifest = json.loads(z.read("manifest.json"))
    db = Path(workdir, "vectors.db"); db.write_bytes(z.read(manifest["database"]))
    embed, sha = manifest["models"]["embed"], manifest["embedding"]["sha256"]
    if isinstance(embed, str):
        # weights are an archive member
        gguf = Path(workdir, "embed.gguf"); gguf.write_bytes(z.read(embed))
    else:
        # weights are a reference by SHA-256
        gguf = Path(models_dir, f"{sha}.gguf")
    assert hashlib.sha256(gguf.read_bytes()).hexdigest() == sha
    # R3
    meta, e = dict(connect(db).execute("SELECT key, value FROM tome_meta")), manifest["embedding"]
    assert [meta[k] for k in ("package_id", "revision", "embedding_id", "embedding_sha256",
                             "dimensions", "metric")] == [manifest["id"], str(manifest["revision"]), e["id"],
                                                         sha, str(e["dimensions"]), e["metric"]]
    # R4
    return manifest, db, gguf

def connect(db):
    con = sqlite3.connect(f"file:{db}?mode=ro", uri=True)
    con.enable_load_extension(True); sqlite_vec.load(con)
    return con

def search(db, vector, k):
    # R5
    con = connect(db)
    rows = con.execute(
        "SELECT c.id FROM vec_chunks AS v JOIN chunks AS c ON c.id = v.rowid "
        "WHERE v.embedding MATCH ? AND k = ? ORDER BY v.distance",
        (struct.pack(f"<{len(vector)}f", *vector), k),
    ).fetchall()
    return [r[0] for r in rows]

```

Figure 4: The core of the independent reader, verbatim: archive rules (R1), checksums (R2), the embedder bound by hash (R3), tome_meta against the manifest (R4), and the exact k -NN query on a read-only connection (R5). The rest of the file starts llama-server on the verified GGUF and embeds each question through POST /v1/embeddings.

- com/pgvector/pgvector, 2021.
- [18] KitOps. KitOps: Packaging and versioning for AI/ML projects as OCI artifacts. <https://github.com/kitops-ml/kitops>, 2024.
 - [19] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.
 - [20] Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.
 - [21] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822. Association for Computational Linguistics, 2023.
 - [22] David Mezzetti and NeuML. txtai: All-in-one AI framework for semantic search and LLM workflows. <https://github.com/neuml/txtai>, 2020.
 - [23] Mintplex Labs. AnythingLLM. <https://github.com/Mintplex-Labs/anything-llm>, 2023.
 - [24] Mozilla.ai. llamafile: Distribute and run LLMs with a single file. <https://github.com/mozilla-ai/llamafile>, 2023.
 - [25] Nomic AI. GPT4All. <https://github.com/nomic-ai/gpt4all>, 2023.
 - [26] Ollama. Ollama: Get up and running with large language models locally. <https://github.com/ollama/ollama>, 2023.
 - [27] Open Container Initiative. OCI image format specification. <https://github.com/opencontainers/image-spec>, 2017.
 - [28] Open WebUI. Open WebUI. <https://github.com/open-webui/open-webui>, 2023.
 - [29] OpenAI. Chat completions API reference. <https://platform.openai.com/docs/api-reference/chat>, 2026. Accessed October 2026.
 - [30] Oded Ovadia, Menachem Brief, Moshik Mishaelli, and Oren Elisha. Fine-tuning or retrieval? Comparing knowledge injection in LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 237–250. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.15.
 - [31] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473. Association for Computational Linguistics, 2019.
 - [32] PKWARE Inc. APPNOTE.TXT – .ZIP file format specification, version 6.3.10. <https://pkware.cachefly.net/>

webdocs/casestudies/APPNOTE.TXT, 2022.

- [33] Qdrant. Snapshots. <https://qdrant.tech/documentation/snapshots/>, 2026. Accessed October 2026.
- [34] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- [35] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muenighoff. C-Pack: Packed resources for general Chinese embeddings. *arXiv preprint arXiv:2309.07597*, 2023. Introduces the BGE embedding models.