WIKIMIND

A research-style case study in large-scale text retrieval, grounded generation and distributed cloud inference

CONSOLIDATED PROJECT STATE / 16 NOVEMBER 2026

Wikipedia RAG | Azure application tier | AWS inference | Tailscale | HTTPS

PUBLIC PROJECT CASE STUDY | LINKEDIN EDITION
Project author: Sayan Sarkar
Consolidated history: V1 → V2 → V3
Reporting cut-off: 16 November 2026
Website: https://www.sanzx.tech

ABSTRACT

Abstract

WikiMind is a web-based retrieval-augmented generation (RAG) application that answers natural-language questions using evidence retrieved from a large offline Wikipedia text corpus. The project integrates large-scale text preparation, SQLite FTS5 search, sentence embeddings, hybrid retrieval, passage diversity, a compact Qwen model served by Ollama, a FastAPI backend, a Next.js interface and a distributed cloud deployment. Its production architecture separates the knowledge/application tier from model inference: the application and retrieval stack run on Azure, while inference runs on an AWS EC2 CPU instance connected through a private Tailscale network. Browser access is protected by HTTPS on the confirmed public hostname.

The source archive was an English no-picture Wikipedia ZIM snapshot of approximately 49.07 GB. The archive audit reported 19,191,219 article entries. A 10,000-entry extraction pilot completed in approximately 2.23 minutes. The production SQLite FTS5 index was reported to contain approximately 7.19 million searchable documents and occupy about 38.7 GiB. A persistent embedding cache supports the semantic retrieval layer.

This case study describes the system as RAG, not as a model training project: the Wikipedia corpus is indexed and searched at question time, while a pre-trained Qwen model generates the final response from selected passages. The project evolved through V1 (first working browser deployment), V2 (AWS inference migration and HTTPS), and V3 (consolidated deployment and documentation state).

Keywords

Retrieval-augmented generation; Wikipedia; ZIM; SQLite FTS5; sentence embeddings; hybrid search; Ollama; Qwen; FastAPI; Next.js; Azure; AWS; Tailscale; HTTPS.

49.07 GB	19.19M	7.19M	5.88 tok/s
Source archive	Article entries audited	Searchable documents reported	One short CPU sample

LIVE PROJECT
Explore the deployed application at https://www.sanzx.tech. The public entry point is presented separately from the private model-inference network.

1 / INTRODUCTION

1. Project motivation and research context

Large language models can formulate fluent responses but do not automatically expose the source passages that support a particular answer. A retrieval-augmented design addresses this by separating the discovery of relevant documents from the generation of a natural-language response. In WikiMind, a question is first matched against a curated Wikipedia-derived corpus. The selected passages and their source metadata are then passed to the language model.

The project objective was to build a working system rather than a model-training pipeline. The Wikipedia snapshot functions as an external knowledge store. SQLite FTS5 supports lexical search; a sentence-transformer model supplies semantic representations; a hybrid ranker combines the signals; and Qwen generates a response from the selected evidence.

1.1 Scope and contributions

- Inspected a large offline English Wikipedia ZIM snapshot and validated extraction APIs before full-scale processing.
- Built a text-cleaning workflow and a persistent SQLite FTS5 index containing millions of searchable records.
- Integrated MiniLM embeddings and a persistent SQLite cache to support semantic matching.
- Implemented hybrid lexical-semantic retrieval and diversity-aware passage selection.
- Built a FastAPI backend with conversation persistence, feedback and Server-Sent Events.
- Developed a Next.js/React interface for chat, conversation navigation, sources and streamed answers.
- Separated application/retrieval infrastructure from model inference using Azure, AWS and Tailscale.
- Added a custom domain and Let’s Encrypt HTTPS for the public browser interface.

1.2 Technical classification

WikiMind is a RAG system. It does not train or fine-tune Qwen on Wikipedia. The data pipeline transforms the source archive into a searchable database. Retrieval happens when the user asks a question; inference happens after evidence selection. This distinction matters because a RAG application, a fine-tuning workflow and a pre-training pipeline have different data requirements, evaluation methods and claims.

2 / DATA AND CORPUS ENGINEERING

2. Source selection and corpus preparation

The project selected `wikipedia_en_all_nopic_2026-06.zim`, an English, no-picture ZIM snapshot with archive metadata dated 17 June 2026. The no-picture flavour supported a text-centred first release and reduced storage and processing requirements compared with an image-bearing archive.

Source audit metric	Recorded value
Archive size	Approximately 49.07 GB.
Article entries	19,191,219.
User entries	19,707,079.
All entries	19,707,096.
Media entries	515,775.
Snapshot metadata date	17 June 2026.

The archive counts are distinct archive categories and should not be interpreted as one cumulative article count. The final indexed-document count is lower than the initial article-entry count. The project record does not provide a full reconciliation of filtering, cleaner exclusions or index insertion outcomes, so the difference is reported without inventing a cause.

2.1 Inspection before extraction

The archive was not processed at full scale immediately. Inspection scripts read metadata, audited entry counts, probed the installed `libzim` interface, tested entry IDs and examined item access. A 1,000-entry benchmark recorded raw HTML size, cleaned text size and text-character counts. Its extrapolation suggested roughly 77.95 GB of uncompressed text, a planning estimate rather than a final measured output size.

2.2 Pilot extraction and cleaning



A 10,000-entry pilot completed in approximately 2.23 minutes and produced a compressed JSON Lines file. Cleaner revisions were tested to convert HTML content into readable text while retaining titles and body content and removing markup and non-content elements where possible. The v3 cleaner reported an error list, but a complete count and taxonomy were not available in the final record.

2.3 Indexed corpus and storage

The final production search database uses an SQLite FTS5 virtual table with `title` and `body` fields. It was reported at approximately 7,194,143 indexed documents and approximately 38.7 GiB (about 41.56 billion bytes). The persistent embedding cache was approximately 58 MB at transfer. FTS5 provides indexed full-text retrieval inside SQLite [1].

> **REPORTED DATA SCALE**
> Source archive: ~49.07 GB. Initial archive audit: 19.19 million article entries. Production FTS index: ~7.19 million records. Estimated uncompressed text scale: ~77.95 GB from a sample benchmark, not a measured final file.

3 / INFORMATION RETRIEVAL

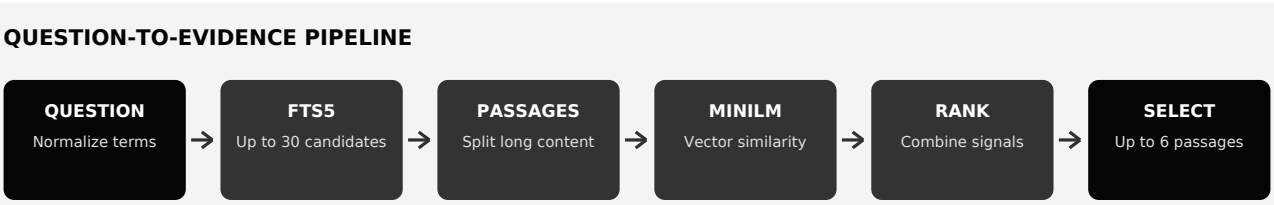
3. Hybrid lexical and semantic retrieval

The retrieval system combines complementary signals. Lexical search is useful where a query contains exact names, phrases, dates or terminology present in the article. Semantic similarity can identify content where the user expresses an idea differently from the source wording. The implementation retrieves candidates lexically and then applies embedding-based similarity and hybrid ranking.

3.1 Search database and semantic encoder

SQLite FTS5 stores article titles and bodies in the `docs` virtual table. The semantic encoder is `sentence-transformers/all-MiniLM-L6-v2`, loaded on the application host. A separate SQLite embedding cache stores reusable passage vectors, reducing repeated encoding work. The model card describes this checkpoint as a sentence-transformer embedding model for semantic similarity tasks [2].

3.2 Retrieval stages



Stage	Recorded implementation
Query preparation	Normalize question and build an FTS expression using up to ten terms.
Lexical candidates	Retrieve up to thirty candidate records from FTS5.
Passage splitting	Divide article content into smaller evidence units.
Embedding	Encode the question and candidate passages; cached passage vectors may be reused; batch size 32 was recorded where applicable.
Hybrid rank	Combine lexical relevance with semantic similarity. Exact rank weights were not available.
Diversity	Select up to six passages and at most two per article.
Source construction	Return title, Wikipedia URL and excerpt capped at roughly 500 characters.

3.3 Observed relevance

A test question about Albert Einstein and relativity returned several directly relevant passages, including articles about relativity theory and its formulations, alongside one weaker result about the given name “Albert”. This is an operational retrieval demonstration, not a formal benchmark. No labelled dataset, recall@k, mean reciprocal rank or citation precision values were recorded.

3.4 Retrieval research agenda

A future evaluation set should include fact-seeking, definitional, multi-hop and ambiguous queries. Each item should have a manually reviewed set of expected article references and relevant passages. Recall@k can assess whether the retriever finds relevant evidence; mean reciprocal rank and nDCG can assess ordering; citation coverage can assess whether generated claims are linked to retrieved source material. The generator should be evaluated separately from the retriever so errors can be attributed to the right component.

4 / GENERATION AND EVIDENCE GROUNDING

4. Model integration and response generation

The model runtime uses Ollama with a Qwen3.5 2B model in Q8_0 quantization. The current production tag is `qwen3.5:2b-q8_0`. The backend supplies the user question and selected Wikipedia evidence to the generation model. Ollama returns streamed content; FastAPI forwards the content to the browser.

4.1 Generation controls

Setting	Recorded current intent
Model	qwen3.5:2b-q8_0.
Quantization	Q8_0.
Streaming	Enabled.
Thinking	Explicitly disabled with `think: false`.
Model keep-alive	30 minutes in the original engine configuration.
Context	A 32,768-token configuration was recorded in the earlier engine.
Output limit	Raised to 1,536 tokens after an earlier response ended near a lower output ceiling.
Temperature	0.1 in the earlier production engine record.

The handler ignores any separate streamed thinking field and forwards only message content. The explicit `think: false` setting is an implementation control, not a claim that the model itself has been modified. Ollama documents the API switch between thinking and direct content output [3].

4.2 Context length versus output length

The input context window governs how much prompt and conversation material can be processed. The output budget constrains the maximum number of generated tokens. The two limits are not interchangeable. A long answer ending abruptly may reflect an output cap, a stop condition, timeout, upstream stream error or context limitation. The observed increase to 1,536 tokens expands the output allowance but cannot guarantee that a very long response will finish in one generation.

4.3 Evidence-grounded generation

The retrieved passages are intended to anchor the answer. Source titles, URLs and excerpts are returned separately so the frontend can show the articles used. RAG makes the answer path more inspectable but does not eliminate hallucinations. The language model can misread a passage, merge facts from different contexts or overstate what the evidence supports. Users should verify important claims against the linked source material.

5 / APPLICATION ENGINEERING

5. Backend, frontend and streaming architecture

The backend is implemented with FastAPI and Uvicorn. It maintains conversation state in a separate SQLite database and connects the application request to retrieval and generation. The browser application is built with Next.js, React and TypeScript. The two application layers are served behind Nginx.

5.1 Conversation data model

Entity	Recorded fields	Role
conversations	id, anonymous_id, title, created_at, updated_at	Conversation ownership scope and timestamps.
messages	id, conversation_id, role, content, sources, feedback, created_at	Chat transcript, sources and feedback.

5.2 API surface

Method	Route	Role
GET	/api/health	Basic service check.
GET / POST	/api/conversations	List and create conversations.
GET / PATCH / DELETE	/api/conversations/{id}	Read, rename and delete a conversation.
POST	/api/messages/{id}/feedback	Record up/down feedback.
POST	/api/chat/stream	Submit a prompt and stream results.

5.3 Server-Sent Events

The streaming protocol uses four named event classes: `sources` for source objects, `token` for a generated text fragment, `done` for completion and message identity, and `error` for message-level failure. The FastAPI response generator emits events as they become available; Nginx disables buffering and caching on `/api/` so the browser can display output progressively. FastAPI documents StreamingResponse as a response class that can stream the contents of a generator or iterator [4].

5.4 Frontend interface

The recorded frontend stack is Next.js 15.5.29, React 19, TypeScript and Tailwind CSS 3.4. The broader dependency set includes Framer Motion 11, Lucide React and React Markdown with remark-gfm. The interface provides a dark responsive layout, a conversation sidebar, search, new chat, a question composer, streamed responses, source metadata, feedback actions and settings controls.

5.5 Browser compatibility correction

The deployed API client was changed from a localhost default to same-origin relative paths so that the browser calls the public website and Nginx routes `/api/` to FastAPI. A separate issue prevented browser API calls in the initial public HTTP context: `crypto.randomUUID()` was unavailable. A compatible fallback was added in both the API helper and chat hook. After correcting a helper typo, the frontend rebuilt successfully, the managed service restarted and the end-to-end browser test passed.

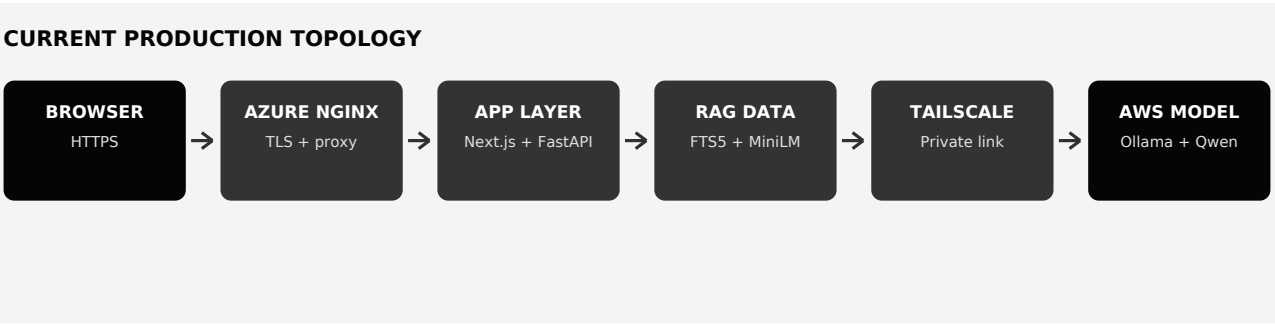
5.6 Anonymous session scope

The browser stores `wm_anon` and `wm_active` in localStorage and sends the anonymous identifier through `X-Anonymous-Id`. This is lightweight session continuity, not a user account or an authorization system. A public multi-user service needs proper authentication and server-side ownership checks.

6 / DISTRIBUTED CLOUD ARCHITECTURE

6. Azure application tier and AWS inference tier

V1 established a split-compute design between an Azure application VM and a Windows desktop running Ollama. That arrangement validated remote inference but kept the public service dependent on the desktop being available. V2 replaced the desktop inference host with AWS EC2. In V3, Azure continues to host the website, FastAPI, the Wikipedia database and embedding cache; AWS hosts Ollama and Qwen; Tailscale provides the private device-to-device route.



6.1 Request lifecycle

- The browser requests the public WikiMind domain over HTTPS.
- Nginx terminates TLS and forwards page requests to Next.js and `/api/` requests to FastAPI on loopback.
- FastAPI creates or resumes the conversation and passes the question to the retrieval engine.
- The engine searches FTS5, obtains cached or newly generated MiniLM embeddings, applies hybrid ranking and selects evidence passages.
- The backend builds the prompt and sends a generation request to Ollama over Tailscale.
- Ollama generates content on AWS CPU; FastAPI converts answer fragments and source metadata into SSE events.
- Nginx streams the events to the browser, where the UI progressively renders the answer.

6.2 V1 to V2 inference transition

Revision	Inference arrangement	Operational implication
V1	Ollama/Qwen on Windows desktop; Azure connected over Tailscale.	Functional but required the personal PC to remain available.
V2 / V3	Ollama 0.40.16 and `qwen3.5:2b-q8_0` on AWS EC2; Azure reaches it over Tailscale.	Cloud-hosted inference removes Windows from the live request path.

6.3 AWS inference host characteristics

The current inference host is an AWS EC2 `m7i-flex.large` instance with 2 vCPU and 8 GiB RAM, running Ubuntu 26.04.1 LTS on x86_64. It has no GPU attached. The Qwen Q8_0 model download was approximately 2.7 GB, with a reported parameter count around 1.9 billion. AWS publishes the M7i-flex instance family as general-purpose CPU compute with the m7i-flex.large size offering 2 vCPU and 8 GiB of memory [5].

6.4 Private inference boundary

Ollama was configured to listen on the AWS Tailscale address at TCP port 11434. The EC2 security group does not expose this port to the public internet. Azure reached the model API over the shared tailnet and successfully retrieved `/api/tags`. Tailscale uses WireGuard-based encryption for device traffic [6]. This design keeps model inference off the public application edge.

6.5 CPU sample result

2 vCPU	8 GiB	~2.7 GB	~5.88
AWS instance	Memory	Q8_0 model	tokens/sec sample

One recorded AWS test took approximately 18.04 seconds in total, including about 7.1 seconds of model load. The generation phase produced about 62 tokens in 10.37 seconds, around 5.88 tokens per second. This is a single small test, not a sustained throughput, concurrency or capacity benchmark.

6.6 Windows desktop status

Ollama remains installed on the original Windows desktop, but it is no longer in the live production request path. It has not been uninstalled. The public application now uses the AWS endpoint; the Windows installation is simply a potential fallback and should not be presented as part of the current production architecture.

7 / DOMAIN, HTTPS AND SECURITY

7. Public delivery and transport security

The V1 website was first exposed over HTTP through the Azure VM public address. The domain and transport layer were then upgraded. The domain now has an A record for the root hostname and a CNAME for the `www` hostname. Certbot 5.10.0 was installed on the Azure VM and used with the Nginx plugin to obtain and deploy a Let’s Encrypt certificate covering both names.

7.1 HTTPS deployment

Certbot successfully deployed the certificate into the Nginx configuration. The reported expiry date is 14 February 2027 and a scheduled renewal task was created. A renewal dry-run has not yet been confirmed. The www hostname has been tested in a browser; WikiMind loaded and the browser showed a secure connection. A direct HTTPS request from the Azure host to the root hostname returned HTTP 200 with Next.js response headers.

7.2 DNS diagnosis

During root-hostname testing, the browser showed an old APIM landing page. Investigation found a resolver-specific discrepancy. The registrar dashboard showed the intended A and CNAME records; all four authoritative nameservers returned the current Azure destination. Google DNS and Cloudflare DNS also returned the correct destination. The local AdGuard resolver continued to return a stale cached address even after the Windows DNS cache was flushed. The `www` hostname resolved correctly and worked over HTTPS.

The important technical distinction is between authoritative DNS and a recursive resolver cache. The authoritative zone can be correct while one resolver continues serving an older cached response until its TTL or cache refresh policy expires. The public case study omits the stale address and all infrastructure IPs.

7.3 Current transport/security boundaries

Layer	Current design	Security / architectural purpose
Browser edge	HTTPS to the confirmed www hostname; Nginx on Azure.	Encrypts browser-to-origin traffic and centralizes public entry.
Application tier	Next.js and FastAPI use loopback upstreams behind Nginx.	Keeps application services behind the reverse proxy.
Knowledge store	SQLite corpus and embedding cache remain on the application tier.	Separates retrieval data from model inference.
Model service	Ollama is reachable through the private Tailscale network.	Avoids exposing the inference API on the public web edge.
Cloud topology	Azure application tier and AWS inference tier.	Allows independent placement of retrieval and generation workloads.

SECURITY POSTURE
HTTPS and private inference networking are implemented. Authentication, server-side authorization, rate limiting, WAF/origin protection, tested backup restoration and formal security testing remain future work. This is a working deployment, not a fully audited multi-user service.

7.4 Security implications

The HTTPS edge protects browser traffic in transit, while the private Tailscale path protects communication between the application host and the inference service. These controls solve different problems and should be considered together. HTTPS does not provide user identity or authorization, and a private model network does not protect public application endpoints from abuse. A hardened public deployment therefore still needs authentication, server-side authorization, request throttling, monitoring, tested backups and a deliberate origin-protection strategy. The current deployment is functional, but those controls should not be represented as completed until implemented and verified.

8 / ENGINEERING RESULTS AND DISCUSSION

8. Verification and observed outcomes

Component	Recorded check	Result / interpretation
Source archive	Metadata and entry audit.	19.19M article entries reported.
Extraction pilot	10,000 entries extracted.	Approximately 2.23 minutes.
FTS5 corpus	Schema and production database.	`docs(title, body)`; ~7.19M docs reported.
Retrieval	Einstein / relativity query.	Several relevant results, one weaker match.
Backend	Health endpoint and dependencies.	FastAPI returned online; runtime imports validated.
Conversation storage	Create, list, read-back.	Persistence path functionally exercised.
SSE stream	Sources, token, completion events.	Streaming path operational.
Private inference	Azure to AWS `/api/tags`.	Ollama model metadata returned across Tailscale.
Model generation	Short Qwen response on AWS CPU.	~5.88 tokens/sec in one generation phase.
Website after migration	User-tested chat flow.	Confirmed working after Azure backend switch.
Public TLS	Browser test on www hostname.	WikiMind loaded with a secure connection.

8.1 Engineering decisions

Problem	Diagnosis	Decision / resolution
Uncertain ZIM APIs	Library and entry semantics needed confirmation.	Created inspection scripts and tested IDs and item access.
Unknown extraction scale	Full extraction without estimates could waste time.	Ran benchmark and pilot first.
Cleaner revisions	HTML text extraction required iteration.	Compared cleaner revisions before indexing at scale.
Slow Azure inference	Observed CPU generation took about 218 seconds.	Moved inference to separate hardware, then AWS.
Windows availability dependency	Personal PC required power and network uptime.	Migrated production Ollama to AWS.
Public browser API errors	localhost base and UUID support blocked frontend calls.	Same-origin API paths and UUID fallback.
SSE proxy buffering	Reverse proxy could delay events.	Disabled buffering/cache for API route.
HTTPS setup	HTTP-only deployment needed TLS.	Certbot Nginx plugin deployed Let’s Encrypt certificate.
Root domain APIM result	AdGuard resolver returned stale destination.	Checked authoritative DNS and independent resolvers; www retained as working hostname.

8.2 Performance interpretation

The AWS model sample indicates that small CPU-only Qwen inference is operational but latency remains a design constraint. The sample should not be used as a service-level objective because it was one short generation, without concurrency or repeated warm-cache measurements. A meaningful evaluation would measure cold-start and warm generation separately, median and tail latency across representative prompts, concurrent user behaviour, CPU/memory saturation and network transit time.

9 / LIMITATIONS, ROADMAP AND REPRODUCIBILITY

9. Limitations and future research

The project demonstrates an end-to-end retrieval and generation system, but several evaluation and operational questions remain open. The corpus count discrepancy should be reconciled, retrieval relevance should be evaluated systematically, the model should be benchmarked under representative load, and the public service needs stronger identity and abuse controls before broader promotion.

9.1 Current limitations

- The difference between the source archive article count and production index count has not been fully reconciled.
- Retrieval weights and labelled ranking metrics were not captured, limiting quantitative claims.
- One weak candidate appeared in the recorded retrieval test.
- A 2B CPU-hosted model has limited throughput and may truncate unusually long reports at the configured output ceiling.
- The browser anonymous ID is not authentication or authorization.
- The root hostname had a resolver-specific stale cache symptom during diagnosis; the www hostname is the confirmed public address.
- Certificate renewal dry-run and external HTTP redirect tests remain to be run.
- No formal penetration test, load test or backup restoration exercise is recorded.

9.2 Prioritized roadmap

Priority	Workstream	Next step
Immediate	TLS and DNS	Run renewal dry-run; test HTTP redirects and root-hostname behaviour after resolver cache refresh.
Immediate	Access control	Introduce real authentication/authorization and harden SSH using keys.
Immediate	Abuse resistance	Add rate limits, request-size controls and input validation.
Immediate	Recovery	Automate backups and test a restore procedure for all SQLite data and configuration.
Near-term	Network perimeter	Review host firewalls and NSGs; assess WAF/proxy deployment.
Near-term	Monitoring	Track health, Ollama availability, CPU, memory, disk, latency and failures.
Near-term	Retrieval quality	Create labelled questions; measure recall@k, MRR/nDCG and citation support.
Later	Scalability	Measure concurrent use, cache hit rate and consider more capable inference hardware only when justified.
Later	Dataset lifecycle	Implement snapshot upgrades with staging, integrity validation and rollback.
Future	Capability expansion	Consider multilingual and multimodal retrieval after text quality and security are stable.

9.3 Reproducibility and evidence discipline

The report separates project-record measurements from formal performance claims. Archive size, archive counts, pilot duration, FTS5 record count, database size, cache size, model version, instance class and the single generation sample are drawn from recorded project outputs. Estimated raw text size is explicitly labelled as an extrapolation. No missing extraction-error count, retrieval score, load-test result or security assurance is invented.

10 / PROJECT EVOLUTION

10. Version history: V1, V2 and V3

Version	Date	Scope and outcome
V1	13 Nov 2026	First working public WikiMind application. Azure hosted database/retrieval/API/frontend; Windows desktop hosted Ollama; private Tailscale path; HTTP-only public website; browser chat confirmed after UUID compatibility correction.
V2	16 Nov 2026	Inference moved from Windows to AWS EC2; Ollama and Qwen Q8_0 deployed on CPU; Azure backend switched to AWS Tailscale endpoint; custom domain configured; Let’s Encrypt certificate installed for root and www; www secure in browser.
V3	16 Nov 2026	Consolidated research-style report integrates full development record and current architecture; distinguishes verified controls from pending hardening and evaluation work.

10.1 Final technical contribution

The main contribution is an integrated, deployable RAG workflow built from a large offline text corpus rather than a hosted search API. It demonstrates the interaction of data preparation, full-text indexing, semantic retrieval, evidence selection, compact-model inference, streaming application delivery and split-cloud operations. The deployment also shows an iterative shift from local inference to cloud-hosted inference while preserving the existing corpus and application tier.

10.2 Public project link

Live website: https://www.sanzx.tech

10.3 Project skills

Area	Technical work represented
Data engineering	ZIM archive audit, Python extraction, HTML cleaning, benchmark design and SQLite index construction.
Search and NLP	FTS5, MiniLM sentence embeddings, persistent embedding cache, semantic similarity and hybrid retrieval.
Generative AI	RAG prompt assembly, Qwen inference through Ollama, generation controls and response streaming.
Web engineering	FastAPI, REST-style endpoints, SSE, SQLite persistence, Next.js, React and TypeScript.
Cloud operations	Azure VM, AWS EC2, Linux, Nginx, systemd, DNS, TLS and private Tailscale connectivity.
Debugging	Archive API reconnaissance, service/process management, browser UUID compatibility, HTTPS and DNS resolver diagnosis.

11 / DETAILED DATA ENGINEERING RECORD

11. Reproducible preparation of the Wikipedia corpus

A key part of the project was making the source data tractable before any question-answering interface was introduced. The archive is a compressed ZIM snapshot, not a ready-to-query relational database. It therefore required an inspection phase, an extraction strategy, HTML-to-text preparation and a separate indexing step. This sequence was intentionally developed with small test scripts before scaling.

11.1 Inspection and measurement sequence

Stage	Activity	Why it mattered
Archive inspection	Read archive metadata and count article/user/media entries.	Established the dataset flavour, scale and archive date.
API probing	Tested entry enumeration, ID handling and item retrieval using libzim.	Reduced uncertainty about the extraction interface.
Sample benchmark	Measured raw HTML bytes, cleaned bytes and text characters across a 1,000-entry sample.	Provided a preliminary storage estimate.
Pilot extraction	Processed 10,000 records into compressed JSON Lines in ~2.23 minutes.	Validated the extraction pipeline at a small but meaningful scale.
Cleaning iteration	Compared cleaner versions and reviewed output/error behaviour.	Improved readable text quality before indexing.
Production indexing	Built the SQLite FTS5 corpus and persistent embedding cache.	Enabled repeated question-time retrieval.

11.2 Interpreting the scale metrics

The archive audit and production index report two different populations. The initial ZIM audit recorded 19,191,219 article entries; the deployed FTS5 database was reported at approximately 7,194,143 documents. The report does not assume that all archive entries are indexable article pages. Redirects, non-article namespaces, unavailable or empty content, cleaner behaviour, duplicates and downstream selection could affect the final indexed population, but the exact contribution of each factor was not measured in the recorded project history. The responsible way to present the result is to label both values and explicitly identify the reconciliation as unfinished.

11.3 Why a text-focused snapshot was appropriate

The first release was scoped around factual text questions with inspectable article sources. The no-picture archive reduced the storage and processing burden while still providing a broad reference corpus. It avoided the need to design an image extraction pipeline, OCR, image embedding, visual retrieval, media deduplication and multimodal prompt handling during the initial milestone. The choice is a scope decision, not a statement that images are unimportant to knowledge systems.

12 / RETRIEVAL DESIGN DETAILS

12. Passage selection as a bridge between search and generation

A full article is often too large and too unfocused to place directly into a compact language-model prompt. WikiMind therefore converts candidate article content into smaller passage units and selects a subset. Passage-level retrieval is the practical bridge between a large indexed corpus and a limited model context.

12.1 Lexical retrieval

SQLite FTS5 allows the application to search indexed title and body text without requiring a separate search cluster. The production query builder extracts a bounded set of terms from the question and uses the FTS index to obtain a limited candidate pool. Lexical retrieval is particularly useful for exact entities, technical names and distinctive expressions. Its limitations include sensitivity to vocabulary mismatch, ambiguity and common words.

12.2 Semantic retrieval

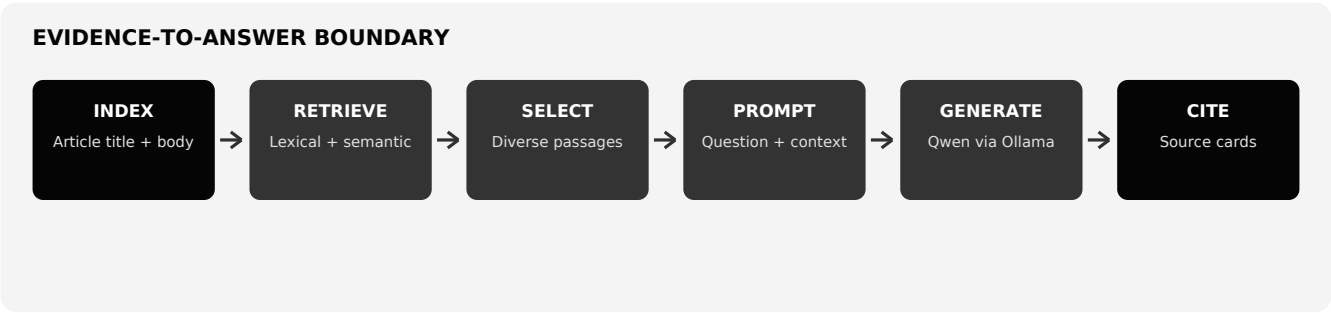
The MiniLM sentence encoder maps questions and passages into vector representations. Similarity between the question vector and passage vectors can surface relevant text even where the wording differs. The persistent cache reduces repeated encoding costs for previously seen passages. The project uses semantic retrieval as a complement to lexical search rather than as a replacement for it.

12.3 Hybrid ranking and passage diversity

The hybrid stage uses both lexical and semantic signals to rank candidate passages. The exact weighting coefficients were not included in the final record and are therefore not reported. The diversity rule limits repeated evidence from a single article to at most two selected passages, with up to six passages in the final evidence set. This helps broaden the evidence bundle, although it cannot ensure that every selected passage is relevant or that all important facets of a question are covered.

12.4 Source metadata and transparency

Each selected source is transformed into a frontend-friendly object with an article title, a Wikipedia URL and an excerpt of approximately 500 characters at most. This design makes it possible for a reader to inspect the originating article. Source visibility is an important transparency feature, but a URL alone does not prove that every generated claim is supported by the cited excerpt. A future citation evaluation should verify claim-to-passage entailment rather than simply counting attached URLs.



13 / APPLICATION CONTRACT AND USER EXPERIENCE

13. Application architecture and interaction model

The application was built as a conventional web stack around the RAG engine. FastAPI provides the HTTP interface and orchestration layer. Next.js supplies the browser application. SQLite stores conversation state independently of the Wikipedia knowledge store. The separation allows the corpus and retrieval logic to remain stable while the inference host changes.

13.1 Conversation lifecycle

A browser creates or resumes a conversation using a browser-scoped anonymous identifier. A question is sent with the conversation identifier and message content. The backend retrieves evidence, starts generation and streams named events back. On completion, the assistant message and selected sources are persisted to the conversation database. The user can later list, open, rename or delete conversation records and submit feedback on an assistant response.

13.2 Event contract

Event	Purpose	Expected UI behaviour
sources	Make selected evidence available before/during answer generation.	Render source cards with article links and snippets.
token	Carry incremental model output.	Append text to the current assistant message.
done	Identify completion and the persisted response.	Stop loading state and finalize message metadata.
error	Carry a request or generation error.	Show a message-level error state.

13.3 Why Server-Sent Events were used

For a text-generation interface, progressive delivery improves the perceived responsiveness of the application because the browser can display content as it arrives. SSE uses a one-way server-to-client stream, which fits this response pattern. Nginx buffering is disabled for the API route so the proxy does not wait to accumulate the whole response. The stream design also separates source metadata, content fragments, completion and errors into discrete event types.

13.4 Frontend compatibility as a deployment concern

The initial public deployment demonstrated that a healthy backend does not guarantee a working browser experience. A helper configured with a localhost URL would cause a remote browser to contact its own device. The code was changed to use same-origin relative URLs, allowing Nginx to route the request. In addition, the browser did not issue API calls when the UUID helper relied on `crypto.randomUUID()` in a non-secure context. A fallback was added in the API and chat-hook code, the production build was regenerated and the managed service restarted. The issue was resolved at the browser layer rather than by changing the RAG database or backend API.

14 / CLOUD DEPLOYMENT AND PROCESS MANAGEMENT

14. Running the application tier as persistent services

The application tier was deployed on an Ubuntu virtual machine in Azure. The recorded VM class was Standard_D4as_v4 with 4 vCPU and 16 GiB RAM, a 256 GB Standard SSD and a configured 4 GB swap file. It hosted Nginx, the Next.js production service, FastAPI/Uvicorn, the SQLite corpus, the embedding cache and conversation storage. Resource names and host addresses are withheld from this public edition.

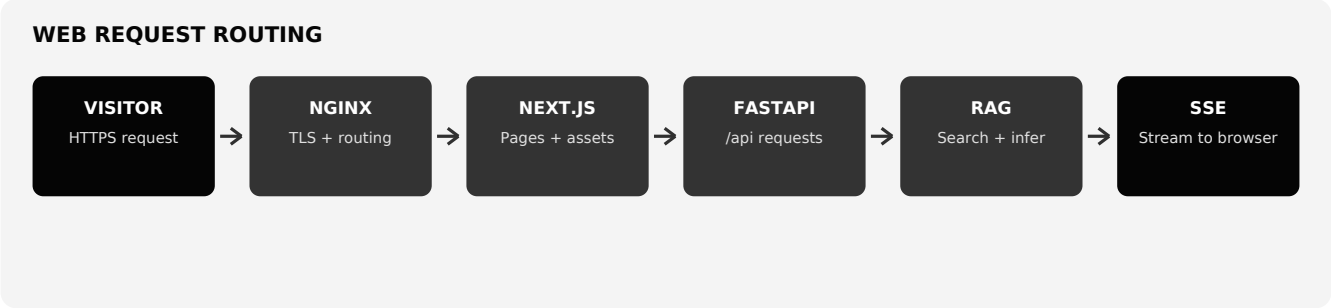
14.1 Service ownership and startup

The frontend and backend were initially launched during setup from interactive terminal sessions. They were later moved into systemd services. The Next.js service executes the production start command from the frontend project directory. The backend service executes Uvicorn from its project virtual environment and loads environment configuration. Nginx is independently managed as the reverse proxy. All services are enabled to start at boot and configured to recover from ordinary process failures.

An early service-start attempt exposed a process ownership conflict: a manually launched Next.js process already occupied the frontend port. The manual process was stopped so that systemd could become the single process manager for the production server. This is a useful operational lesson: do not operate both a manual foreground instance and a systemd-managed instance on the same production port.

14.2 Reverse-proxy routing

Nginx is the public entry point. Page requests and static assets are proxied to the Next.js server; `/api/` requests are proxied to FastAPI. The API location uses HTTP/1.1, disables proxy buffering and caching, and sets a long read timeout to support long-running model responses. The application services remain behind the reverse proxy rather than being independently exposed to the internet.



14.3 Separation of data and application state

The large Wikipedia FTS5 database and the embedding cache are stored separately from the frontend assets. The conversations database is distinct from the read-mostly corpus. This separation makes it possible to rebuild or update the web application without replacing the large knowledge database, provided the deployment process preserves the data directory and configuration.

14.4 Deployment principle

The app tier and inference tier are independently managed. Replacing the inference runtime does not require re-extracting Wikipedia, rebuilding the FTS5 index or moving the conversation database. The Azure `.env` change redirected the model endpoint and tag while the database path remained unchanged. This reduced migration scope and preserved the established knowledge pipeline.

15 / INFERENCE MIGRATION AND RESOURCE TRADE-OFFS

15. From desktop inference to cloud inference

The system went through two inference arrangements before reaching its current cloud-hosted state. The initial Azure-only CPU test took approximately 218 seconds for a response, which was judged too slow for the intended interaction. The V1 implementation therefore used the available Windows desktop as an inference host, with Azure connecting over Tailscale. This improved the architecture but made production availability depend on a personal computer.

15.1 V2 migration to AWS

The production model was moved to an AWS EC2 m7i-flex.large instance with 2 vCPU and 8 GiB of RAM. The host uses Ubuntu 26.04.1 LTS and runs Ollama 0.40.16. The model is `qwen3.5:2b-q8_0`, approximately 2.7 GB in Q8_0 quantization. A GPU was not attached. The selection was a CPU-hosted inference step intended to remove the personal desktop from the serving path; it was not an assertion that the instance offers GPU throughput.

15.2 Measuring the initial CPU result

A short local model test on AWS took about 18.04 seconds including approximately 7.1 seconds to load the model. The model evaluation phase emitted about 62 tokens in 10.37 seconds, which corresponds to approximately 5.88 tokens per second. Since the sample was short and did not test multiple concurrent requests or repeated warm runs, it should be interpreted as a point observation rather than a benchmark of sustained service capacity.

15.3 Operational consequences of the migration

Dimension	V1 desktop inference	Current AWS inference
Availability dependency	Personal PC power, sleep state, network and Ollama process.	Cloud VM and Ollama service availability.
Network path	Private Tailscale connection.	Private Tailscale connection retained.
Compute	Desktop hardware.	2 vCPU / 8 GiB CPU-only instance.
Application changes	Initial model endpoint pointed to desktop.	Backend environment changed to AWS Tailscale endpoint and current model tag.
Knowledge data	Azure database and retrieval.	Unchanged on Azure.

15.4 Migration integrity

The backend environment was backed up before the endpoint change. Only the Ollama base URL and model tag were changed; the FTS5 database path was left in place. The backend service was restarted and became active. Azure successfully retrieved model tags from AWS over Tailscale, and the website was used after the migration. Ollama remains installed on the original desktop as a possible fallback, but that machine is not a current production dependency.

16 / DOMAIN, TLS AND PUBLIC RESOLUTION

16. Domain configuration and HTTPS deployment

The initial V1 website used a public HTTP entry point. The V2 update attached a custom domain and configured TLS. Certbot 5.10.0 was installed on the Azure host and used the Nginx plugin to obtain and deploy a Let’s Encrypt certificate for the root and www hostnames. The certificate was reported to expire on 14 February 2027, and Certbot installed a scheduled renewal task. The renewal dry-run remains to be confirmed.

16.1 Nginx TLS deployment

The HTTPS virtual host listens on the secure web port and references the full certificate chain and private key managed by Let’s Encrypt. It preserves the original reverse-proxy mapping: `/api/` to FastAPI and `/` to Next.js. The HTTP server block contains host-specific redirect rules to HTTPS and a fallback 404. The presence of the fallback response does not itself mean that the HTTPS site is returning 404; scheme and hostname must be tested separately.

16.2 DNS resolution as a layered system

The public domain used an A record for the root hostname and a CNAME alias for the www hostname. During troubleshooting, the registrar dashboard showed only the intended website records. The four authoritative nameservers returned the correct Azure destination; Google DNS and Cloudflare DNS returned the same current address. One recursive resolver continued returning an outdated answer for the root hostname even after the local Windows DNS cache was flushed. The www hostname was confirmed working in the browser over HTTPS.

This investigation distinguished authoritative records from recursive caching. A stale response in one recursive resolver does not mean that the authoritative zone is misconfigured. It also demonstrated the practical value of testing with several resolvers and querying authoritative nameservers directly.

16.3 What HTTPS does and does not provide

HTTPS protects browser-to-origin traffic in transit and allows the browser to validate the hostname against a certificate. It does not conceal the origin address when public DNS points directly to the host. Hiding the origin would require a proxy/CDN arrangement and careful origin firewall rules. No WAF or Cloudflare proxy is claimed as part of the current deployment.

> **PUBLIC HOSTNAME**
> The confirmed public address for this case study is https://www.sanzx.tech. The root hostname experienced a stale resolver response during the recorded troubleshooting window; the authoritative DNS zone itself returned the configured Azure destination.

17 / SECURITY AND TRUST MODEL

17. A layered security assessment

Security controls exist at several different boundaries and should not be conflated. HTTPS protects the browser connection. Nginx centralizes public web ingress. Loopback binding keeps the application processes behind the proxy. Tailscale protects the private Azure-to-AWS path. The application still needs identity, authorization, abuse resistance and recovery controls before it should be treated as a hardened multi-user service.

17.1 Implemented boundaries

Layer	Implemented design	Security purpose
Browser transport	Let’s Encrypt TLS certificate for both hostnames; www confirmed secure.	Encrypts the browser session and validates the hostname.
Web edge	Nginx as the sole public web reverse proxy.	Centralizes routing and TLS termination.
Application services	Next.js and FastAPI use loopback upstreams.	Avoids direct public exposure of internal app ports.
Private inference	Tailscale path from Azure to AWS.	Keeps model requests off the public web path.
Ollama listener	Bound to the private overlay interface.	Prevents treating the model API as a public endpoint.
Data separation	Corpus/cache/conversation stores outside the public frontend.	Separates knowledge files and state from static web assets.

17.2 Controls still needed

Control	Status in recorded project	Recommended action
Authentication	Anonymous browser identifier only.	Add user accounts or trusted identity integration.
Authorization	No full server-side ownership model demonstrated.	Enforce ownership checks for every conversation operation.
Rate limiting	Not recorded as implemented.	Limit requests per identity/IP and protect expensive generation paths.
WAF / origin shielding	Proxy/WAF layer not completed.	Evaluate a managed edge/WAF and restrict origin access only after safe testing.
Backup recovery	No restore drill recorded.	Back up index, cache, conversations and configuration; rehearse restoration.
SSH hardening	Password login was used during cloud setup.	Move to tested key-based login and restrict administrative sources.
Monitoring	Basic health endpoint and service logs.	Add alerts for model downtime, latency, disk, memory and error rates.
Formal testing	No penetration or load-test report.	Commission a scoped security and performance assessment.

17.3 Risk interpretation

The application should be described as a functioning, HTTPS-enabled prototype with private inference connectivity. It should not be described as fully secure, formally audited or suitable for storing sensitive personal information. This distinction keeps the public project description technically accurate while giving a clear direction for the next development phase.

18 / VALIDATION, RESEARCH LIMITATIONS AND FUTURE WORK

18. Verification matrix and evaluation plan

Area	Observed evidence	Interpretation
Source archive	Metadata, article count and archive flavour inspected.	Initial data scale verified.
Extraction	10,000-entry pilot completed in ~2.23 minutes.	Extraction path validated on pilot.
Search corpus	FTS5 schema and ~7.19M index documents reported.	Production search store created.
Semantic layer	MiniLM loaded and persistent cache available.	Semantic retrieval integrated.
Retrieval query	Einstein/relativity results included several useful sources and one weak match.	Functional retrieval; quality needs systematic evaluation.
Backend health	Health endpoint returned online.	API process responded.
Conversation persistence	Create/list/read checks performed.	Conversation storage exercised.
Streaming	Sources, token and completion events observed.	Progressive response path works.
Tailscale inference path	Azure read the AWS Ollama model list.	Private route verified.
Generation sample	~5.88 tokens/sec for one short CPU sample.	A single observed speed, not an SLA.
Public website	User tested chat after migration.	Current request path functionally verified.
HTTPS	www hostname loaded securely in browser.	Public TLS verified for confirmed hostname.

18.1 Evaluation questions for the next research iteration

- Does lexical retrieval find expected articles for exact entity and terminology queries?
- Does semantic retrieval improve recall for paraphrased questions compared with FTS alone?
- Does diversity selection improve source coverage without excluding the strongest evidence?
- How often does each generated claim have support in the displayed source excerpts?
- What is the end-to-end latency distribution for cold and warm model runs?
- How does the system behave under multiple simultaneous users?
- How often do long answers stop because of output limits, timeouts or upstream errors?

18.2 Proposed evaluation design

A labelled test collection should contain several question categories: entity lookup, definition, timeline, comparison, causal explanation, multi-hop reasoning and ambiguous phrasing. For each query, reviewers should mark relevant articles and passages. Retrieval should be measured independently using recall@k, precision@k, reciprocal rank and nDCG. Generated answers should be assessed for factual support, completeness, source attribution and unsupported statements. Latency testing should separate model-load time from warm generation and network overhead.

18.3 Reliability test plan

A reproducible reliability run should include repeated queries across short and long prompts, a cold start, a warm start, an Ollama restart, a temporary tailnet interruption, a backend restart and a browser refresh. Each test should capture logs, model completion status, final response persistence and whether the source objects were retained. This plan would turn the current functional validation into a repeatable acceptance suite.

19 / REPRODUCIBILITY AND TECHNICAL INVENTORY

19. Reproducibility notes and component inventory

Layer	Public technical description
Source data	English no-picture Wikipedia ZIM snapshot, approximately 49.07 GB.
Index	SQLite FTS5, `docs(title, body)`, approximately 7.19 million records reported.
Embedding model	sentence-transformers/all-MiniLM-L6-v2.
Embedding cache	Separate SQLite database for reusable passage representations.
Retrieval implementation	Python query normalization, FTS candidate retrieval, passage splitting, hybrid rank and diversity selection.
Generation	Ollama 0.40.16 with Qwen3.5 2B Q8_0.
Backend	FastAPI / Uvicorn with conversation persistence and SSE.
Frontend	Next.js 15.5.29, React 19, TypeScript and Tailwind CSS 3.4.
Application hosting	Azure Ubuntu VM, 4 vCPU and 16 GiB RAM recorded.
Inference hosting	AWS EC2 m7i-flex.large, Ubuntu 26.04.1, 2 vCPU and 8 GiB RAM.
Private networking	Tailscale between the application and inference hosts.
Browser transport	Custom domain with Let’s Encrypt HTTPS.

19.1 Configuration separation

The backend stores its model endpoint and model tag in environment configuration rather than hard-coding the model server into retrieval code. This allowed the V1 desktop host to be replaced by AWS without modifying the corpus, retrieval algorithms or conversation database. In future deployment variants, the inference endpoint can be changed while keeping the public application tier stable.

19.2 Data protection and change management

The project preserved the large SQLite corpus and embedding cache while changing application settings and service configuration. This is an important operational principle: configuration and application builds should be treated separately from persistent knowledge data. A safe deployment process should retain a known-good database copy, back up environment files before editing, validate Nginx syntax before reload, rebuild the frontend before service restart, and test the health and chat paths after the change.

19.3 Project learning outcomes

The work required understanding a compressed knowledge archive, converting it into searchable records, combining exact-term search with semantic representations, choosing evidence for a compact model context, handling streamed generation, exposing APIs to a browser, resolving reverse-proxy and browser compatibility issues, moving inference to a cloud host and validating DNS and TLS. It demonstrates the practical coupling between data engineering, retrieval, generative AI, full-stack software and infrastructure operations.

20 / CONCLUSION

20. Conclusion

WikiMind demonstrates an end-to-end approach to building a knowledge assistant around a large offline Wikipedia corpus. Its contribution is not a newly trained foundation model, but an application architecture that combines a persistent searchable dataset, lexical and semantic retrieval, passage selection, evidence-grounded generation, a browser interface and a split-cloud deployment.

The V1 milestone delivered the first working public chat system, with the Azure application and retrieval tier connected to a Windows-hosted Ollama runtime. V2 improved availability by moving inference to AWS EC2 and strengthened the public access layer by adding a custom domain and HTTPS. V3 consolidates those engineering phases and presents the system as it was verified on 16 November 2026.

The principal technical lessons are that retrieval quality must be evaluated independently of generation; output-token limits must not be confused with context-window limits; the public browser layer can fail independently of healthy backend services; TLS and DNS resolver behaviour must be tested separately; and a private inference service should remain off public network interfaces. The next phase should focus on quantitative evaluation, user identity, abuse protection, backup recovery, monitoring and reliability testing.

The live project can be viewed at https://www.sanzx.tech. The public hostname is retained in this document; private infrastructure identifiers and operational addresses are deliberately excluded.

REFERENCES

Selected technical references

- [1] SQLite Project. “FTS5 Extension.” https://www.sqlite.org/fts5.html (accessed 16 Nov 2026).
- [2] Sentence Transformers. “all-MiniLM-L6-v2 Model Card.” https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 (accessed 16 Nov 2026).
- [3] Ollama. “Thinking.” https://ollama.com/blog/thinking (accessed 16 Nov 2026).
- [4] FastAPI. “Custom Response - StreamingResponse.” https://fastapi.tiangolo.com/advanced/custom-response/ (accessed 16 Nov 2026).
- [5] Amazon Web Services. “Amazon EC2 M7i Instances.” https://aws.amazon.com/ec2/instance-types/m7i/ (accessed 16 Nov 2026).
- [6] Tailscale. “Tailscale Encryption.” https://tailscale.com/docs/concepts/tailscale-encryption (accessed 16 Nov 2026).
- [7] Ubuntu Server Documentation. “Obtain TLS Certificates.” https://ubuntu.com/server/docs/how-to/security/obtain-tls-certificates/ (accessed 16 Nov 2026).
- [8] Microsoft Learn. “Azure Network Security Groups Overview.” https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview (accessed 16 Nov 2026).
- [9] Next.js. “Deploying.” https://nextjs.org/docs/app/getting-started/deploying (accessed 16 Nov 2026).

AUTHORSHIP AND ACKNOWLEDGEMENT

WikiMind was independently designed, assembled, tested and deployed. AI-assisted tools supported code review, debugging, command planning and documentation refinement. The project decisions, corpus work, service configuration, deployment and verification were carried out through the project development process.