Zimbabwe Weather Intelligence: An Evidence-Based Climate Research Assistant and One-Hour-Ahead Temperature Prediction System

Rosemary Mtape

MacroEdTech – Part 02 Project

November 2026

Abstract

Climate information is important for understanding weather variability, assessing climate-related risks and supporting decisions in climate-sensitive sectors. However, relevant information about Zimbabwe's climate is often distributed across reports and other documents, while weather forecasting and climate research require different forms of data processing and analysis. This project extends the Zimbabwe Weather Intelligence platform by developing a generative artificial intelligence (GenAI) component that combines document-based climate question answering with access to weather prediction functionality.

The implemented system provides a Research Assistant for answering questions about Zimbabwe's weather and climate and a separate Temperature Prediction interface for estimating temperature one hour ahead in Harare, Bulawayo, Mutare and Gweru. The research component uses a retrieval-augmented generation (RAG) approach: source documents are divided into text chunks, converted into embeddings and indexed for semantic retrieval using FAISS. Retrieved context is supplied to a language model to support evidence-based responses. LangChain and LangGraph are used in the agent workflow, while a Model Context Protocol (MCP) server exposes selected weather API functions as tools. The temperature prediction component reuses the project's machine-learning prediction API.

The knowledge base contains 217 indexed chunks. The underlying weather dataset, collected from Open-Meteo for the period 2015–2025, contains 385,728 records. The XGBoost one-hour-ahead temperature model achieved a mean absolute error (MAE) of 0.3448 on the reported evaluation. The system was first implemented locally using Ollama and was subsequently adapted for browser-based cloud access

using the Gemini API. The cloud Temperature Prediction functionality was tested successfully after the separately hosted Render API service had been activated. The cloud Research Assistant returned relevant information for several test questions but subsequently became temporarily unavailable for further generation requests after the applicable Gemini free-tier quota was exhausted. The local Ollama implementation continued to operate. This paper describes the system architecture, implementation approach, available evaluation evidence and deployment trade-offs. It also discusses limitations, including dependence on the quality and coverage of source documents, the possibility of incomplete generated answers, and the external API quota and availability constraints introduced by the cloud deployment.

Keywords: Zimbabwe; weather intelligence; climate research; generative artificial intelligence; retrieval-augmented generation; FAISS; LangGraph; Model Context Protocol; XGBoost; temperature prediction.

1 Introduction

Weather and climate information support decisions in agriculture, water management, disaster preparedness, public planning and other climate-sensitive activities. In Zimbabwe, understanding historical climate events and accessing relevant evidence can help users investigate questions about drought, agricultural impacts, climate variability and related topics. At the same time, access to a machine-learning-based temperature prediction capability can provide a complementary view of near-term weather conditions.

The Zimbabwe Weather Intelligence project addresses these needs through a software platform that combines weather data analysis and temperature prediction. Part 02 extends the platform with a generative AI research capability. The extension is intended to make selected climate information easier to query using natural language while retaining access to the project's existing temperature prediction service.

The system has two principal user-facing functions. First, the Research Assistant accepts natural-language questions about Zimbabwe's weather and climate. Document-based questions are handled through an evidence-first retrieval workflow, while general weather questions can be routed through an agent workflow that has access to selected weather API tools. Second, the Temperature Prediction interface allows a user to provide weather measurements and request a one-hour-ahead temperature estimate for a supported city.

This paper documents the design and implementation of the Part 02 system. It explains the retrieval-augmented generation workflow, the use of vector search and language models, the integration of the weather prediction API through MCP, and the deployment adaptation from a local Ollama implementation to a cloud-accessible Gemini implementation.

1.1 Problem statement

Climate-related information may be distributed across reports and other documents that are not easily searched through ordinary keyword queries. Users may need to read several documents to find evidence relevant to a particular question. In addition, a document question-answering system alone does not provide access to live or API-mediated weather functions, while a standalone prediction interface does not support broader questions about historical climate evidence.

The project therefore addresses the need for an integrated interface that supports both document-grounded climate research and access to a separate one-hour-ahead temperature prediction capability. The system must also make its functions accessible through a browser-based application while retaining a local implementation path.

1.2 Aim and objectives

The overall aim was to extend the Zimbabwe Weather Intelligence platform with a GenAI-powered climate research assistant and an integrated interface for one-hour-ahead temperature prediction.

The specific objectives were to:

1. prepare a searchable knowledge base from selected Zimbabwe weather and climate documents;
2. implement a retrieval-augmented generation workflow for natural-language climate questions;
3. use a vector database to retrieve semantically relevant document passages;
4. implement an agent workflow capable of accessing selected weather API functions through MCP tools;
5. integrate the existing temperature prediction service into a user-facing application;
6. test the principal application functions and document the observed results and limitations; and
7. deploy a browser-accessible version while retaining the local Ollama implementation.

1.3 Scope

The system focuses on evidence-based questions about Zimbabwe's weather and climate and one-hour-ahead temperature prediction for Harare, Bulawayo, Mutare and Gweru. It is not presented as a replacement for official meteorological warnings or professional

climate analysis. The temperature model predicts temperature only; it does not directly predict rainfall probability, thunderstorms or other weather variables.

2 Background and Related Work

2.1 Retrieval-augmented generation

Retrieval-augmented generation combines information retrieval with text generation. Rather than relying only on information encoded in a language model's parameters, a RAG system retrieves relevant external passages and supplies them as context to the model when generating a response. Lewis et al. describe this approach as a way of combining parametric language-model memory with non-parametric retrieved knowledge [1].

A typical RAG workflow includes document ingestion, text extraction, chunking, embedding generation, vector indexing, similarity retrieval and context-conditioned answer generation. The approach is relevant to this project because answers to Zimbabwe-specific climate questions should be informed by the selected source documents rather than generated solely from general model knowledge.

RAG does not guarantee that every answer is complete or correct. Its performance depends on the quality of the source documents, the relevance of retrieved passages, chunking decisions, embedding quality and the language model's ability to use the retrieved context appropriately.

2.2 Vector similarity search and FAISS

Text embeddings represent text as numerical vectors that can be compared in a vector space. Semantic retrieval uses these representations to identify passages that are conceptually related to a query, even when the query and passage do not use exactly the same words.

FAISS is a library for efficient similarity search and clustering of dense vectors [2]. In this project, FAISS stores the embedded knowledge-base chunks and supports retrieval of passages relevant to a user's question.

2.3 Language-model agents and workflow orchestration

Language-model agents can use tools to obtain information or perform operations beyond text generation. Workflow orchestration frameworks provide mechanisms for defining how an application processes a request, selects tools and passes information between steps. LangGraph is used in the project to structure the general weather-agent workflow. Its role is distinct from the document retrieval process: the RAG workflow retrieves evidence from

the indexed knowledge base, whereas the agent workflow can use selected API tools when a request requires weather-service functionality.

2.4 Model Context Protocol

The Model Context Protocol (MCP) provides a standardised way for AI applications to connect to external tools and data sources. The project uses an MCP server to expose selected functions from the weather prediction API. This allows an MCP-compatible client or agent to discover and invoke supported functions through a defined tool interface [4].

2.5 Gradient-boosted decision trees

XGBoost is an implementation of gradient-boosted decision trees designed for scalable and efficient machine learning [3]. The existing Zimbabwe Weather Intelligence prediction component uses an XGBoost model for one-hour-ahead temperature prediction. Part 02 integrates this existing capability rather than treating the GenAI language model as the temperature forecasting model.

3 System Architecture and Design

3.1 Overall architecture

The Part 02 application presents two principal functions through a Streamlit user interface: the Research Assistant and Temperature Prediction. The research function distinguishes document-based climate questions from general weather questions. Document-based questions use the RAG knowledge base, while general weather questions are handled through the LangGraph agent and its available MCP tools. The prediction interface sends user-provided meteorological measurements to the existing prediction API.

The main architectural components are:

- **Streamlit user interface:** provides the browser-based application and its navigation tabs.
- **Knowledge-base pipeline:** extracts and prepares source-document text, divides it into chunks and creates embeddings.
- **FAISS vector index:** stores vectors and supports similarity-based retrieval.
- **Language-model layer:** generates responses using retrieved context for document-based questions.
- **LangGraph agent:** orchestrates general weather requests and tool interactions.

- **MCP server:** exposes selected weather API operations as tools.
- **Weather prediction API:** provides access to the existing XGBoost temperature prediction service.

3.2 Research Assistant interface

Figure 1 shows the Research Assistant interface. It provides a natural-language question input and explains that document-based climate questions use evidence-first research functions, while general weather questions use the LangGraph agent and MCP tools. This distinction is important because the two request paths rely on different information sources and processing steps.

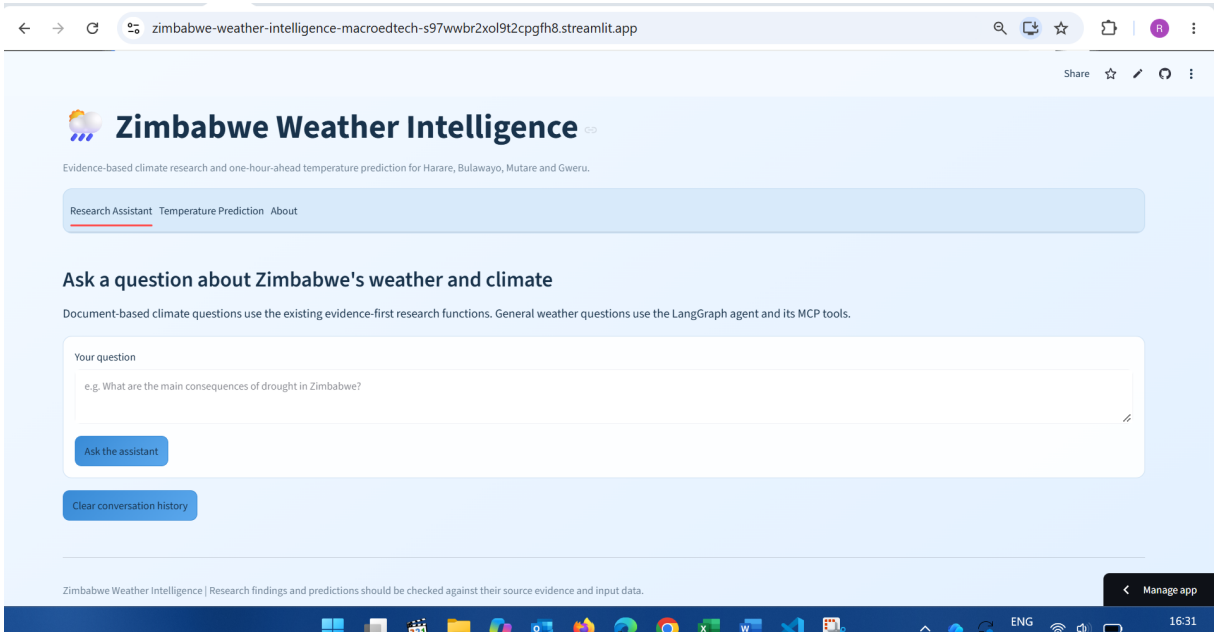


Figure 1: Research Assistant interface of the Zimbabwe Weather Intelligence application.

3.3 Temperature Prediction interface

Figure 2 shows the Temperature Prediction interface. Users select a supported city and provide the input date, time and meteorological measurements. The displayed inputs include temperature, humidity, precipitation, atmospheric pressure, wind speed, wind direction and cloud cover. The interface also asks users to confirm the accuracy of the measurements before requesting a prediction.

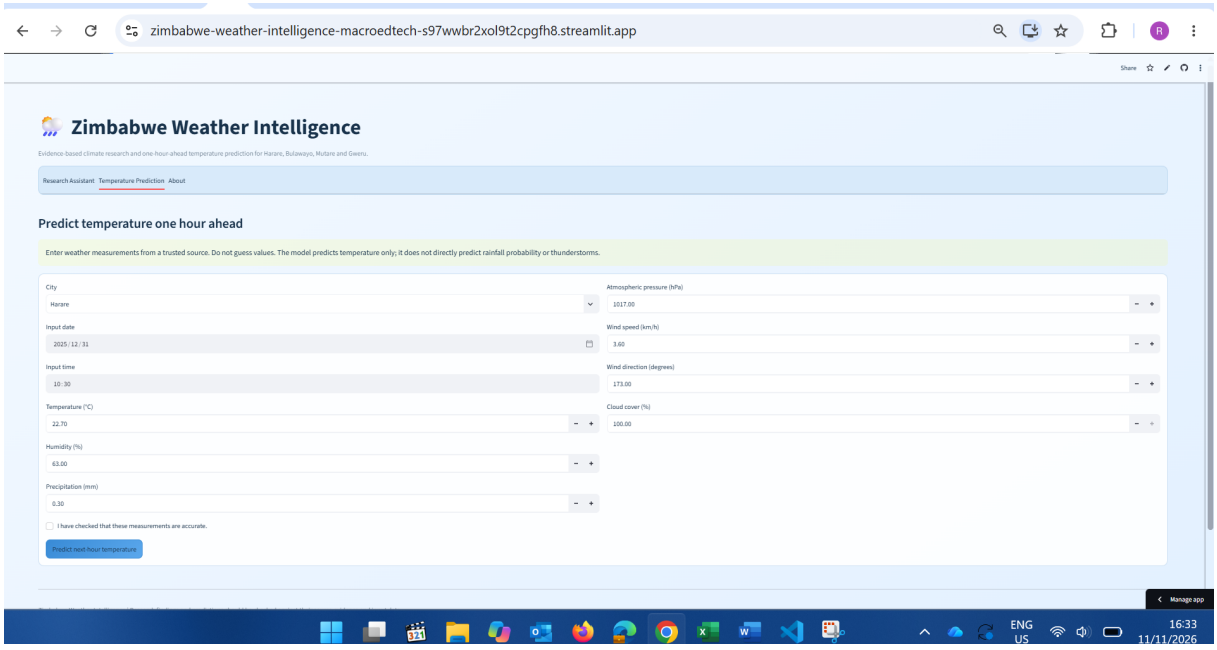


Figure 2: Temperature Prediction interface showing the meteorological input fields used to request a one-hour-ahead temperature estimate.

3.4 Knowledge-base and retrieval architecture

The document-based research workflow follows these general stages:

1. **Document preparation:** selected climate-related source documents are collected for processing.
2. **Text extraction:** textual content is extracted from the source files.
3. **Chunking:** extracted text is divided into smaller passages suitable for embedding and retrieval.
4. **Embedding:** each chunk is converted into a vector representation.
5. **Indexing:** the vectors are stored in a FAISS index.
6. **Retrieval:** a user's question is embedded and compared with indexed vectors to identify relevant passages.
7. **Generation:** retrieved passages are supplied to the language model as context for response generation.

The deployed Gemini-based index contains 217 indexed chunks. The index is stored in the project under `part02_genai/data/faiss_index_gemini`, with the FAISS index and its associated metadata files.

3.5 Weather API and MCP integration

The MCP server is implemented in `part02_genai/src/weather_mcp_server.py`. It exposes the following API-related tools:

- `get_weather_api_health` – checks the availability or health of the weather API;
- `get_available_cities` – retrieves the cities supported by the service; and
- `predict_temperature` – submits weather measurements to the prediction endpoint.

The supported cities are Harare, Bulawayo, Mutare and Gweru. The MCP server provides a tool interface to the API. The Temperature Prediction tab also provides a direct user-facing route to the prediction service. These are related but distinct integration paths. The availability of a tool on the MCP server does not, by itself, demonstrate that every deployed agent request automatically discovers and invokes that tool.

4 Methodology

4.1 Development approach

The project followed an iterative implementation and testing approach. The existing weather prediction service from Part 01 was retained, while the GenAI research and agent capabilities were developed as an additional component. The implementation was tested locally before a browser-accessible cloud deployment was prepared.

4.2 Weather dataset and prediction component

The underlying weather dataset was obtained from Open-Meteo and covers the period 2015–2025. The dataset contains 385,728 records. The existing Part 01 modelling workflow uses XGBoost to predict temperature one hour ahead. The reported model evaluation result is a mean absolute error (MAE) of 0.3448.

Mean absolute error measures the average absolute difference between observed and predicted values:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|,$$

where y_i is the observed temperature, $\hat{y}_i$ is the predicted temperature and n is the number of evaluated observations.

The reported MAE is 0.3448. The unit should be stated as degrees Celsius if confirmed against the prediction target and evaluation code. The paper does not infer additional performance metrics from this single result.

4.3 Knowledge-base preparation

Selected climate-related documents were processed to create the searchable knowledge base. The extracted text was divided into chunks, embedded and indexed using FAISS. The deployed Gemini-based knowledge base contains 217 chunks and uses the embedding model `models/gemini-embedding-001`.

The retrieval process is designed to provide relevant source passages to the language model before answer generation. The generated answer may still be incomplete if relevant evidence is absent from the knowledge base, not retrieved, or not adequately incorporated into the response.

4.4 Language models and local implementation

The initial local implementation used Ollama to run the `llama3.2:3b` language model and `nomic-embed-text` embedding model. This approach allowed the system to be developed and tested locally without sending model prompts to a hosted Gemini API.

The local implementation remains part of the project and provides a route for local execution. Its practical resource requirements depend on the machine running the models, including available memory, storage and processing capacity.

4.5 Cloud deployment adaptation

For browser-based access, the deployed application was adapted to use the Gemini API for language generation and embeddings. Hosting the Ollama models and their downloaded weights alongside the cloud application would require additional storage, memory and compute resources. Using the hosted Gemini API reduced the need to run and store those models on the application server and made it practical to provide supervisor access through a web browser.

This was a deployment adaptation rather than a claim that the deployed system is entirely open-source or independent of external services. The Gemini-based deployment introduces dependencies on internet connectivity, API-key management, provider availability, request quotas and any applicable API charges. The local Ollama implementation is retained separately.

4.6 Application and integration testing

Testing focused on whether the principal application components could be accessed and whether the implemented workflows returned usable outputs. The available project test evidence includes:

- successful loading and use of the deployed Streamlit application;

- successful use of the Temperature Prediction interface with the deployed prediction API after the Render service had been activated;
- creation and use of a FAISS index containing 217 chunks;
- smoke tests of climate questions concerning the 1991/92 drought, the 2015/16 drought, fall armyworm, annual crop losses and the AFOLU sector;
- successful testing of the MCP server and its three exposed API tools in the integration workflow; and
- successful operation of the local Ollama implementation during local testing.

The testing was functional and exploratory. It should not be interpreted as a formal, statistically controlled evaluation of answer accuracy, retrieval precision or user satisfaction. No numerical RAG accuracy score is reported because a labelled question-and-answer benchmark and scoring procedure have not been established in the evidence available for this paper.

5 Implementation

5.1 Application interface

The Streamlit application is organised into three navigation areas: Research Assistant, Temperature Prediction and About. The Research Assistant accepts a user's question and routes it to the relevant research or agent functionality. The Temperature Prediction tab collects the required meteorological inputs and submits them for one-hour-ahead prediction.

5.2 Technology stack

Table 1 summarises the principal technologies used in the Part 02 implementation.

Table 1: Principal technologies used in Part 02.

Technology	**Role in the system**
Python	Main implementation language.
Streamlit	Browser-based user interface.
LangChain	Components for language-model and retrieval integration.
LangGraph	Orchestration of the general weather-agent workflow.
FAISS	Vector indexing and similarity retrieval for the knowledge base.
Ollama	Local execution of the initial language and embedding models.
Llama 3.2:3b	Local language model used in the initial implementation.
nomic-embed-text	Local embedding model used in the initial implementation.
Gemini API	Hosted language generation and embedding services in the cloud deployment.
FastAPI	API service for the existing weather prediction functionality.
MCP	Tool interface exposing selected weather API functions.
XGBoost	Machine-learning model for one-hour-ahead temperature prediction.
Open-Meteo	Source of the underlying 2015–2025 weather dataset.

5.3 Deployment and Access Instructions

The Part 02 Streamlit application is deployed using Streamlit Community Cloud. Its application entry point is `part02_genai/src/weather_assistant_ui.py`. The weather prediction API is hosted separately on Render. The deployed application communicates with this API when temperature prediction functionality is requested.

The following links provide access to the deployed components:

1. **Weather Prediction API:**
   `https://zimbabwe-weather-intelligence-macroedtech.onrender.com/`
2. **Zimbabwe Weather Intelligence User Interface:**
   `https://zimbabwe-weather-intelligence-macroedtech-s97wwbr2xol9t2cpgfh8.streamlit.app/`

This browser-based deployment was prepared to allow the project supervisor and other authorised reviewers to access and interact with the application without installing the local language models or configuring an Ollama environment on their own computers.

5.3.1 Instructions for Accessing and Testing the Application

The weather prediction API is hosted on Render, which may place the service into an inactive state after a period without requests. Consequently, the first request after inactivity may experience a delay while the service starts.

To test the Temperature Prediction functionality, users are advised to:

1. open the Weather Prediction API link provided above;
2. wait for the API homepage to return a response indicating that the service is running;
3. open the Zimbabwe Weather Intelligence user interface;
4. navigate to the Temperature Prediction tab;
5. select a supported city, enter the required meteorological measurements and submit the prediction request.

Opening the API link first helps initialise the service before using the prediction interface. However, the API homepage response confirms that the service is responding; it does not independently verify that the prediction endpoint is functioning correctly. Successful submission of a prediction request through the Temperature Prediction interface provides the relevant functional test of that endpoint.

5.3.2 Cloud Research Assistant Availability and API Quota

The Research Assistant uses the Gemini API for language generation and embeddings in the cloud deployment. Its availability is therefore subject to the applicable Gemini API rate limits and quotas. During testing, the free-tier generation quota for the configured model was exhausted, resulting in a `429 RESOURCE_EXHAUSTED` response. The error reported a daily limit of 20 generation requests for the project and model at the time of testing. This figure describes the limit reported during this test and should not be interpreted as a permanent limit applicable to every Gemini model or account.

When the applicable quota is exhausted, the cloud Research Assistant may be temporarily unable to generate further responses until the quota becomes available again. Users should consult Google's current rate-limit information for the applicable model, project and account. Current usage and applicable limits can be monitored through the Google AI Studio rate-limit dashboard:

`https://ai.dev/rate-limit`

The quota restriction affected the cloud Research Assistant's ability to process further questions during the affected testing period. It did not indicate a failure of the local Ollama implementation, which continued to operate on the developer's machine. The cloud Temperature Prediction functionality was tested separately and returned predictions successfully after the Render API service had been activated.

5.3.3 Local and Cloud Deployment Configurations

The cloud deployment requires the Gemini API key to be configured as a deployment secret rather than committed to the source-code repository. This is an important security practice because API credentials should not be exposed in public source code.

The local Ollama implementation remains operational on the developer's machine and is retained as the local execution version. It uses the `llama3.2:3b` language model and `nomic-embed-text` embedding model. The cloud application uses the Gemini API for language generation and embeddings so that the supervisor can access the system remotely through a browser. These are two deployment configurations of the project: the local version runs open-weight models through Ollama, while the publicly accessible version relies on Google's hosted Gemini services. The latter therefore has external-service dependencies, including internet connectivity, API-key management, request quotas, provider availability and any applicable usage charges.

6 Results and Discussion

6.1 Knowledge-base and research-query results

The deployed knowledge base contains 217 indexed chunks. Smoke tests were conducted using questions on several Zimbabwe climate topics. These included the 1991/92 and 2015/16 droughts, fall armyworm, annual crop losses and the AFOLU sector.

The tested AFOLU question returned information indicating an approximate 54% contribution in 2017, and described changes in the sector's net carbon balance across the periods covered by the indexed source material. The response also identified drivers discussed in the retrieved evidence, including deforestation, fuelwood use, livestock, logging and veld fires. This demonstrates that the system can retrieve and synthesise information on a topic represented in its knowledge base. The reported value and interpretation should be checked against the original source document when used for policy or scientific decision-making.

A test concerning the 2015/16 drought returned relevant information, including a reported figure relating to cattle deaths associated with livestock disease and a declining maize-yield trend. However, the answer was not fully comprehensive. This illustrates an important limitation: relevant retrieval and a coherent generated response do not necessarily mean that all relevant evidence has been retrieved or included.

6.2 Cloud Research Assistant Availability During Testing

During testing, the cloud-deployed Research Assistant temporarily became unable to process further questions because the Gemini API free-tier generation quota had been

exhausted. The application returned a `429 RESOURCE_EXHAUSTED` response, with an error message reporting a daily limit of 20 generation requests for the configured project and model at the time of testing. Consequently, further cloud Research Assistant requests could not be completed until the applicable quota became available again.

This limitation affected the cloud Research Assistant's language-generation functionality and did not indicate a failure of the local Ollama implementation. The local version, using `llama3.2:3b` and `nomic-embed-text`, continued to operate successfully. The deployed Temperature Prediction functionality was tested separately and returned predictions successfully through the Streamlit interface after the Render API service had been activated. The API homepage response was used to initialise the Render service; successful prediction requests through the application provided the functional test of the prediction endpoint.

6.3 Temperature prediction results

The existing XGBoost model achieved a reported one-hour-ahead temperature prediction MAE of 0.3448 on the project's evaluation. The Temperature Prediction interface was also tested against the deployed prediction API and returned predictions successfully.

The MAE provides a quantitative indication of the model's average absolute prediction error on the evaluated data. It should be interpreted in the context of the evaluation dataset, target units, time period and validation design. A full assessment should also report the train-test strategy, sample size of the evaluation set, baseline performance and additional metrics where available.

6.4 MCP integration

The MCP server exposes three weather API tools: API health checking, retrieval of supported cities and temperature prediction. The server and its tools were tested as part of the integration work. The deployed interface also provides direct access to the temperature prediction API through its prediction tab.

A distinction is maintained between successful testing of the MCP server tools and confirmation that the deployed general-purpose research agent independently selects every available tool for every relevant query. In particular, automatic discovery and invocation of the temperature prediction MCP tool by the deployed research agent require further verification.

6.5 Deployment trade-offs

The transition from a local Ollama implementation to a Gemini-backed cloud deployment improved the practicality of browser-based access without requiring the supervisor to install and run local language models. However, this introduces an external service dependency.

The cloud version is affected by provider availability, network access, request quotas and possible usage charges.

The two implementations therefore serve different purposes. The local Ollama implementation supports local execution with open-weight models, subject to local hardware requirements. The Gemini-backed deployment supports easier remote access but relies on a hosted commercial API. This distinction is relevant when considering reproducibility, long-term operating costs, data governance and the original preference for open-source and locally deployable components.

7 Limitations and Future Work

The project has several limitations that should guide future development.

1. **Knowledge-base coverage:** the quality of document-based answers is limited by the relevance, accuracy, recency and coverage of the indexed documents.
2. **Formal RAG evaluation:** a labelled evaluation dataset has not yet been used to calculate retrieval precision, recall, answer faithfulness or answer completeness.
3. **External API dependency:** the Gemini deployment requires internet access and is subject to model- and account-specific API quotas, provider availability and applicable charges. The 20-request daily limit reported during testing should not be assumed to apply universally or permanently.
4. **Agent-tool verification:** further tests are required to establish whether the deployed agent reliably discovers and invokes each MCP tool, including the temperature prediction tool.
5. **Operational resilience:** the separately hosted prediction API may experience cold-start delays after periods of inactivity, which can affect the responsiveness of the first request.

Future work should prioritise improving and expanding the climate knowledge base, providing source references in generated answers, and developing a repeatable RAG evaluation dataset. Tool-selection tests should be added for each MCP function. The prediction component could be evaluated across cities, seasons and different validation periods. Deployment documentation should also describe API key handling, quotas, costs and the steps required to run the local Ollama version.

8 Conclusion

This project extended the Zimbabwe Weather Intelligence platform with a GenAI-based climate Research Assistant and a user-facing one-hour-ahead temperature prediction interface. The research component uses retrieval-augmented generation, FAISS vector search and language-model generation to support natural-language questions about Zimbabwe's weather and climate. A LangGraph agent and MCP server provide an additional route for general weather requests and access to selected weather API functions. The prediction interface reuses the existing XGBoost-based weather prediction service.

The implementation produced a searchable knowledge base containing 217 chunks and retained the existing weather dataset of 385,728 records covering 2015–2025. The reported XGBoost model MAE was 0.3448. Functional tests demonstrated that the deployed application could be accessed and that the Temperature Prediction interface could communicate with the prediction API after the Render service had been activated. The Research Assistant returned relevant information for several tested climate topics during testing, although subsequent cloud requests were temporarily restricted when the Gemini free-tier generation quota was exhausted. The local Ollama implementation continued to operate successfully. Some generated answers were incomplete, demonstrating the need for source verification and more systematic evaluation.

The project also demonstrated a practical distinction between local model execution and cloud deployment. The local Ollama implementation remains available, while the browser-accessible deployment uses the Gemini API to reduce the infrastructure required to host language and embedding models. This improves remote accessibility but introduces external API dependencies and quota limitations. Overall, Part 02 establishes an integrated foundation for evidence-based climate information retrieval and short-term temperature prediction, while identifying clear priorities for improving reliability, evaluation and deployment sustainability.

Acknowledgements

The author acknowledges the use of Open-Meteo weather data and the open-source software libraries and frameworks that supported the development of the project. Further institutional or supervisory acknowledgements may be added where appropriate.

References

[1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020. Available: `https://arxiv.org/abs/2005.11401`.

[2] J. Johnson, M. Douze and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019. doi: https://doi.org/10.1109/TBDATA.2019.2921572.

[3] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794. doi: https://doi.org/10.1145/2939672.2939785.

[4] Model Context Protocol, "Specification," Available: https://modelcontextprotocol.io/specification. Accessed: November 2026.

[5] Open-Meteo, "Open-Meteo Weather API," Available: https://open-meteo.com/. Accessed: November 2026.

[6] Streamlit, "Streamlit Documentation," Available: https://docs.streamlit.io/. Accessed: November 2026.

[7] LangChain, "LangGraph Documentation," Available: https://docs.langchain.com/oss/python/langgraph/overview. Accessed: November 2026.

[8] Meta AI, "Faiss: A library for efficient similarity search and clustering of dense vectors," Available: https://github.com/facebookresearch/faiss. Accessed: November 2026.