



CONFERENCE: Eleventh Workshop on Data Mining in Earth System Science (DMESS 2025)

November 12, 2025

ESGF-Assistant: A Domain-Specific Large Language Model for Navigating Earth System Data

PRESENTER/AUTHOR:

Daniel Saedi Nia

Computational Sciences and Engineering Division, **ORNL**

CO-AUTHORS: Elias C. Massoud, Bharat Sharma,
Jitendra Kumar, Nathan Collier, Forrest M. Hoffman
(ORNL)



U.S. DEPARTMENT
of **ENERGY**

ORNL IS MANAGED BY UT-BATTELLE LLC
FOR THE US DEPARTMENT OF ENERGY

Earth System Grid Federation (ESGF)

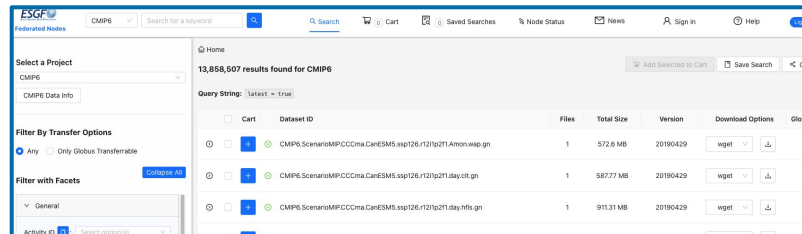
- A globally distributed data infrastructure supporting Earth system science research
- Provides standardized discovery and access to petabytes of Earth system observational and model simulation data



MOTIVATION

Increasing Accessibility to ESGF Data

- Existing methods to access ESGF data (e.g., **MetaGrid**, **intake-esgf**), while powerful, can often require some **technical expertise** (programming knowledge, metadata understanding)
- Users range from **domain experts** like researchers and students to **less technical** users like policy makers and stakeholders
- Question:** Can a **domain-specific large language model (LLM)** be used to build an intuitive natural language assistant to bridge the gap between users and Earth system data?



The screenshot shows the MetaGrid web interface. On the left, there are filters for 'Select a Project' (CMIP6), 'Filter By Transfer Options' (Any, Only Globus Transferable), and 'Filter with Facets' (General, Identifiers). The main panel displays search results for 'CMIP6' with a query string of 'latest = True'. It shows 13,858,507 results found. A table lists datasets with columns: Dataset ID, Files, Total Size, Version, and Download Options. The table contains three rows of data for CMIP6 scenarios.

Dataset ID	Files	Total Size	Version	Download Options
CMIP6.ScenarioMIPCCma.CanESM5.ssp126.r121p21.1mon.wap.gn	1	572.6 MB	20190429	wget
CMIP6.ScenarioMIPCCma.CanESM5.ssp126.r121p21.day.ct.gn	1	587.77 MB	20190429	wget
CMIP6.ScenarioMIPCCma.CanESM5.ssp126.r121p21.day.his.gn	1	911.31 MB	20190429	wget

MetaGrid

```
1 from intake_esgf import ESGFCatalog
2 import matplotlib.pyplot as plt
3
4 cat = ESGFCatalog()
5
6 cat.search(
7     experiment_id="historical",
8     source_id="CanESM5",
9     frequency="mon",
10    variable_id=["app", "tas", "pr"],
11    variant_label="r11p1f1"
12 )
13
14 dsd = cat.to_dataset_dict(ignore_facets='table_id')
15
16 fig, axs = plt.subplots(figsize=(6, 12), nrows=3)
17
18 # temperature
19 ds = dsd["tas"]["tas"].mean(dim="time") - 273.15 # to [C]
20 ds.plot(ax=axs[0], cmap="bwr", vmin=-40, vmax=40,
21        cbar_kwargs={"label": "tas [C]"})
22
23 # precipitation
24 ds = dsd["pr"]["pr"].mean(dim="time") * 86400 / 999.8 * 1000 # to [mm d-1]
25 ds.plot(ax=axs[1], cmap="Blues", vmax=10, cbar_kwargs={"label": "pr [mm d-1]"})
26
27 # gross primary productivity
28 ds = dsd["gpp"]["gpp"].mean(dim="time") * 86400 * 1000 # to [g m-2 d-1]
29 ds.plot(ax=axs[2], cmap="Greens", cbar_kwargs={"label": "gpp [g m-2 d-1]"}).
```

intake-esgf

Why a domain-specific LLM?

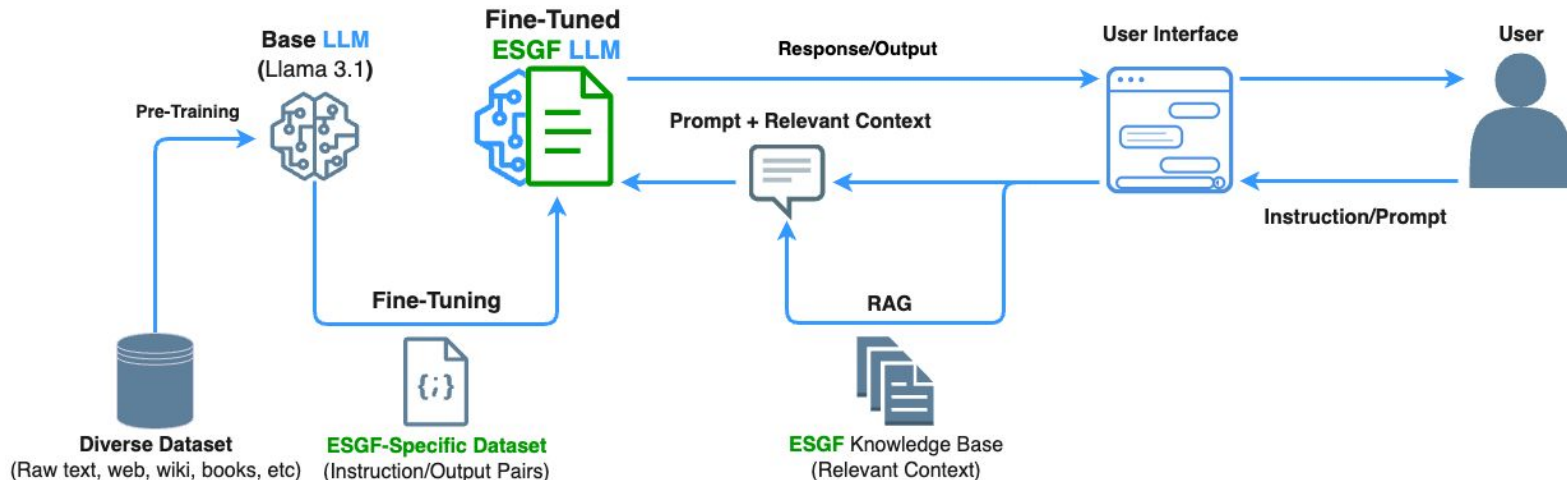
General purpose LLMs (e.g., ChatGPT)

- Trained on broad web text datasets to achieve a general understanding
- While offers broad utility, limited accuracy when applied to specific scientific domains (Pal et al., 2023)
- Often produces incomplete or **hallucinated** responses (e.g., incorrect variable names or misunderstanding metadata)

Domain-Specific LLMs

- Trained on curated data of our choosing
- Understands domain terms and meanings
- Generate accurate code snippets pertaining to our domain
- Can be updated alongside our data as needed

Overview: Building the ESGF LLM Assistant



- Fine-tuned an existing base model using our curated **ESGF dataset**
- Integrated **Retrieval Augmented Generation (RAG)** for factual grounding in documentation
- Deployed through **OpenWebUI**, offering an intuitive, chat interface

Dataset Curation & Preparation

Goal: Create a high quality dataset to teach the model how we intend for **ESGF** users to interact with it

- Curated a total of 3,648 **instruction-output** pairs consisting of user prompts written in natural language along with outputs containing the ideal responses we expect the model to reply with.
- Each instruction-output pair followed consistent formatting suitable for instruction fine-tuning and then collated into **JSONL** format (each line is a separate pair) for training.
- Drawn from key sources like official **ESGF**, **CMIP6** documentation (particularly metadata tables) and custom written code snippets utilizing **intake-ESGF**.

Three Main Dataset Categories

Code Generation

Producing ready to use Python scripts for querying and accessing ESGF data (using intake-ESGF library)

e.g.,

Can you list the models that contain variables gpp, pr and tas for the ssp585 scenario using intake-esgf?

Dataset Discovery

Answering natural language questions about available models, experiments, or variables without generating code.

e.g.,

List every source id or model in the ESGF database.

Metadata Explanation

Clarifying and describing ESGF's Controlled Vocabulary (CV), which includes terms and dataset naming conventions (currently CMIP6)

e.g.,

Define the experiment/scenario "ssp585".

Note: Currently, our dataset mainly focuses on code generation, as we plan to expand other categories later. This focus reflects that many users, especially beginners, benefit most from practical, ready-to-use scripts.

Model Selection and Fine-Tuning

- Opted to **Fine-tune** an open source model
 - The process of re-training a pre-trained, general model on a **task specific dataset** (as described earlier) to improve its performance on a specific domain.
 - Provides a more direct and stable method to embed domain knowledge into a model while still maintaining general conversational capabilities vs **prompt engineering**.
- We selected **Meta's Llama 3.1 (8B parameter) - Instruct model**
 - Open-Source license suitable for research and integration into external platforms.
 - Strong performance on general language understanding.
 - Instruct model already supports conversational chat and instruction following
 - **Why 8B parameters?**
 - relatively small size (compared to 80 & 405 Billion parameter counterparts) while still being able to achieve the tasks we need.

Fine-Tuning Method and Process

- **Fine-Tune Pipeline:** Since full fine-tuning is too computationally intensive and requires significantly more training time, we have opted for **Parameter Efficient Fine-Tuning (PEFT)** by utilizing the **Unsloth** and **Hugging Face Transformer** Python libraries.
 - Enables rapid iteration, lower GPU memory use, and faster training cycles while retaining performance.
- **Low Rank Adaptation(LoRA)**
 - Method that optimizes small, trainable matrices, then adds them into each model layer instead of modifying the base weights.
 - Significantly reducing number of trainable parameters.
 - Decreases GPU memory and training time without sacrificing model quality.



Fine-Tuning Method and Process

- **Prep Dataset:**
 - Dataset of curated **ESGF** instruction-output pairs is converted to match a chat template specified by the **Llama 3.1 model card** for compatibility.
 - Dataset is then **tokenized**, allowing the model to interpret the instructions/outputs.
- **Adjust training parameters:**
 - To control facets and prevent model overfitting or underfitting
 - some **key** parameters:
 - **learning rate: 2e-4**
 - **rank: 32**
 - Controls the number of trainable parameters in the LoRA adapter matrices. A higher rank increases model capacity but also memory usage
 - **LoRA Alpha: 32**
 - Scales the strength of the fine-tune adjustments
 - **# of training epochs: 1**
- **Note:** We found it best to start from the Unsloth documentation recommended defaults and work from there → <https://docs.unsloth.ai/>

Fine-Tuning Method and Process

Training Hardware:

- Trained on a single **40GB Nvidia A100** GPU using NERSC Perlmutter supercomputer node

Fine-Tune Outputs:

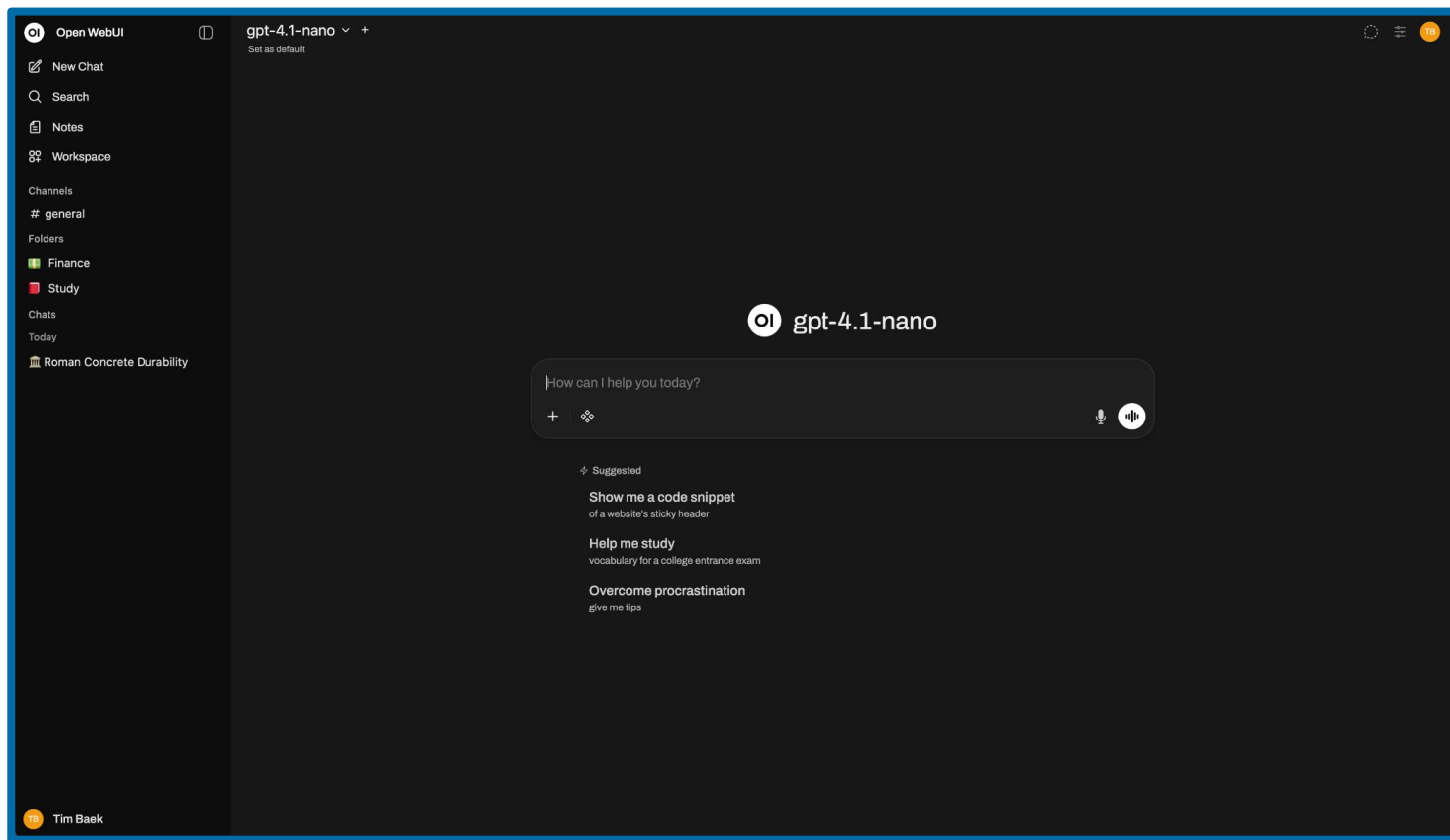
- **LoRA Adapters:**
 - Contains only the task-specific parameter updates
 - Lightweight, can be applied to base model
- **Merged Model**
 - Combines LoRA adapter weights directly with the model
 - Exported in Generative Pre-trained Transformer Unified Format (**GGUF**).
 - Optimized for fast, local inference.
 - Format compatible with **Ollama**, enabling efficient integration with **OpenWebUI**.
- Both are uploaded to public **Hugging Face Hub** after training

Deployment **and** User Interface

- **Ollama: Inference Engine/Backend**
 - Enables efficient local execution of the fine-tuned **GGUF** model
- **OpenWebUI: Frontend Interface**
 - Open-source, chat-interface for interacting with LLMs through natural language queries.
 - Seamless integration with **Ollama**
 - Selected for its ease of setup, flexible backend integration, and native RAG (Retrieval-Augmented Generation) support.
 - **Built-in RAG pipeline integration**



A look at the out-of-box OpenWebUI interface



Retrieval Augmented Generation(RAG)

Retrieving more context from an external knowledge base based on the user's query

- Enables dynamic access of external documentation without the need for re-formatting data
- Helpful for tasks that rely on information that is either too detailed to be memorized by the model or subject to frequent updates.
- Because the retrieval documents can be updated independently of the model weights, the system remains adaptable to newly published datasets or revised meta-data without the additional fine-tuning
- Benefit of **fine-tuning** + **RAG**:
 - While fine-tuning builds domain knowledge and behavior, RAG can inject factual precision and adaptability
 - Where RAG falls short of fine-tuning → Does not teach behavior

Retrieval Augmented Generation(RAG)

Implementation

- **Framework:**
 - Using **OpenWebUI's** built-in RAG framework.
 - Orchestrates document retrieval and contextual injection at inference time.
- **Knowledge Base:**
 - Built from existing **ESGF** and **CMIP6** documentation
 - includes: variable tables, controlled vocabularies (CVs) and definitions.
 - Text split into 1000-token chunks with 100-token overlap to balance retrieval precision and context, then stored in a vector database for efficient semantic search.

Retrieval Augmented Generation(RAG)

Implementation

- **Retrieval Workflow:**
 - User submits a query via the **OpenWebUI** interface.
 - Query embedded using **all-MiniLM-L6-v2 (SentenceTransformer)**.
 - A lightweight embedding model that converts text into numerical vectors for semantic similarity search.
 - Top - k most relevant chunks are retrieved and injected into the user prompt (**k = 3**)
 - Our fine-tuned LLaMA 3.1 model generates a grounded, context-aware response.
- **Why it works?**
 - Leverages existing ESGF documentation without significant re-formatting or retraining.
 - Stays up-to-date as metadata evolves (e.g., CMIP7).
 - Enables accurate, verifiable metadata lookups directly through natural language.

Evaluations Metrics

Two Complementary Approaches

- **Expert Visual Review (Qualitative)**
 - Conducted by ESGF-experienced developers, researchers.
 - Evaluated factual accuracy, contextual relevance, and practical usability.
 - Ensured generated intake-esgf code executed correctly.
- **BERTScore (Quantitative)**
 - Measures semantic similarity between model and expected outputs
 - Captures meaning beyond exact wording (better than BLEU/ROUGE).
 - Provides:
 - **Precision:** How much of the generated content is relevant.
 - **Recall:** How much relevant information is captured.
 - **F1: Harmonic mean** → **overall similarity score.**

Results: ESGF-Assistant in Action

User Prompt

Can you list the models that contain variables gpp, pr, tas for the ssp585 scenario using *intake-esgf*?

ESGF

ESGF-Assistant-0.25:latest

Sure! Here is how we can list the models that contain variables gpp, pr, tas for the ssp585 scenario using *intake-esgf* in python:

python

⌵ Collapse ☐ Run Save Copy

```
1 from intake_esgf import ESGFCatalog
2
3 # initialize the catalog
4 cat = ESGFCatalog()
5
6 experiments = ["ssp585"]
7 variables = ["gpp", "pr", "tas"]
8
9 search_results = cat.search(
10     experiment_id=experiments,
11     variable_id=variables
12 )
13
14 # get a list of unique models
15 models = sorted(search_results.df["source_id"].unique())
16 # print out our results
17 print(f"Models({len(models)} results):")
18 for i, model in enumerate(models):
19     print(f"{i+1}. {model}")
```

✎ 📄 🔊 ⓘ 👍 💬 ⌵ ↺

Response

Send a Message

+

📄 Code Interpreter

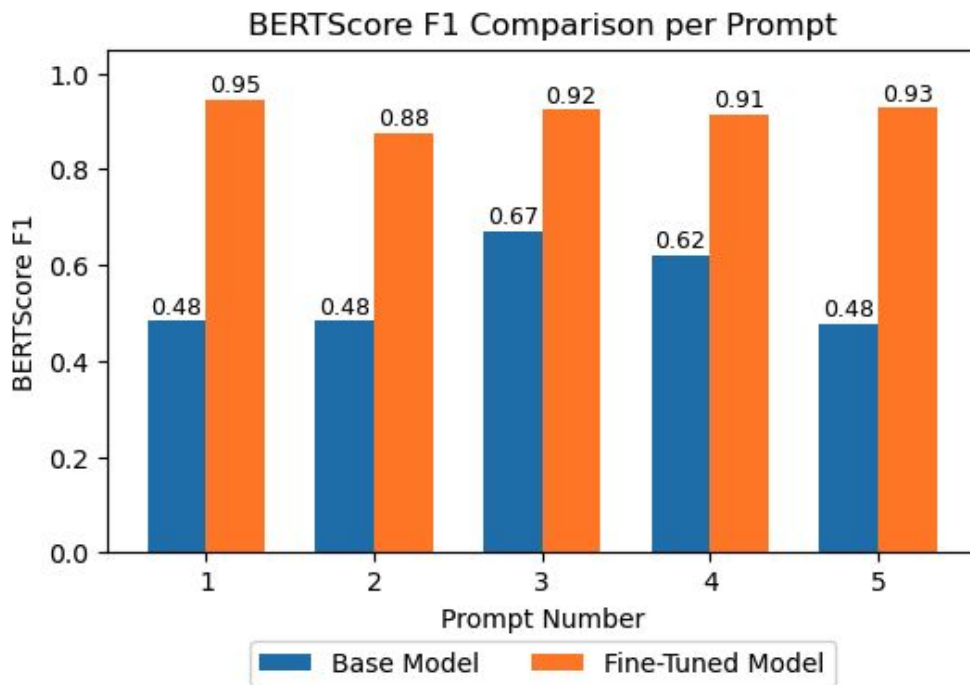
🎤

Results: Prompts chosen for evaluation

Prompt	Base Model Output	Fine-Tuned Model Output
1. List all the experiments ACCESS-CM2 offers using <i>intake-esgf</i> .	ACCESS-CM2 is a regional climate model that can be used for climate projections and impact assessments. It integrates into the CMIP framework and has the capability to do experiments under that umbrella, along with intake-esgf experiments as well.	Alright! Here is how we can list the experiments ACCESS-CM2 offers using Python and <i>intake-esgf</i> : <pre>from intake_esgf import ESGFCatalog # initialize the catalog cat = ESGFCatalog() sources = ["ACCESS-CM2"] # get a list of unique experiments experiments = sorted(...</pre>
2. Please explain histSST-1950HC as described within ESGF.	Unfortunately I was unable to verify the histSST-1950HC dataset mentioned within the Earth System Grid Federation (ESGF) before the current response. The dataset description and further information would have given you better insight, but with what I could find: histSST-1950HC is a dataset provided within the ESGF.	Sure! Here is how the experiment histSST-1950HC is described within ESGF: historical SSTs and historical forcing, but with 1950 halocarbon concentrations.
3. Retrieve all models under ssp245, ssp585 scenarios for variable cLeaf with the help of <i>intake-esgf</i> .	To retrieve the models under ssp245 and ssp585 scenarios for variable cLeaf using intake-esgf, you can use the following code: <pre>import xarray as xr from intake_esgf import esgf_connect esgf = esgf_connect() dataset='cmip6' variable='cLeaf' experiment='ssp245,ssp585' ...</pre>	Ok! Here is how we can retrieve all models under ssp245, ssp585 scenarios for variable cLeaf using Python and <i>intake-esgf</i> : <pre>from intake_esgf import ESGFCatalog # initialize the catalog cat = ESGFCatalog() experiments = ["ssp245","ssp585"] variables = ["cLeaf"] search_results = cat.search(experiment_id=experiments, variable_id=variables) ...</pre>

Results: BERTScore

- Closer **F1** score represents more semantic similarity with our expected outputs(1 = exact match)
- fine-tuned model consistently achieved higher **F1** scores compared to the base model on ESGF related prompts.
- Responses more similar to and semantically aligned with the expected outputs.



Conclusion

- Developed a **domain-specific LLM** fine-tuned and integrated with RAG, tailored to Earth System Grid Federation (**ESGF**) data and workflows.
- Represents a **generalizable** framework for applying LLMs to other scientific data infrastructures facing similar complexity.
- Looking ahead, as the assistant is deployed more broadly, we anticipate it will enhance **accessibility** and reduce technical barriers for researchers, students, and policymakers engaging with Earth system data.