

WORKSHOP

ALCF Hands-on HPC Workshop

October 7-9, 2025

Argonne National Laboratory



U.S. DEPARTMENT
of ENERGY

Argonne
NATIONAL LABORATORY



DEPLOYING AI INFERENCE SERVICES AT ALCF

BENOIT CÔTÉ
Data Services
Software Developer
ALCF
bcote@anl.gov

ADITYA TANIKANTI
Data Services
Architect
ALCF
atanikanti@anl.gov

TOM URAM
Data Services and
Workflows Team Lead
ALCF
turam@anl.gov

VENKATRAM VISHWANATH
Data Science
Team Lead
ALCF
venkat@anl.gov



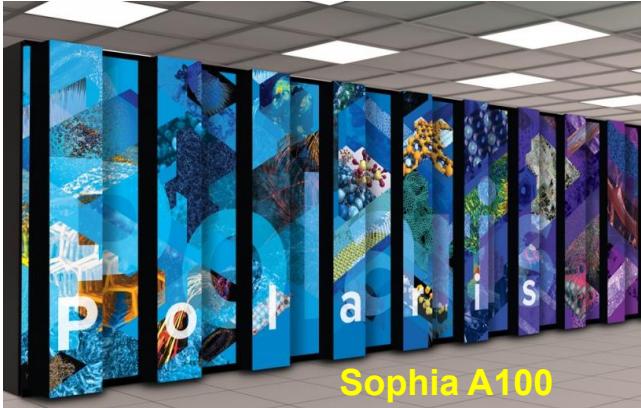
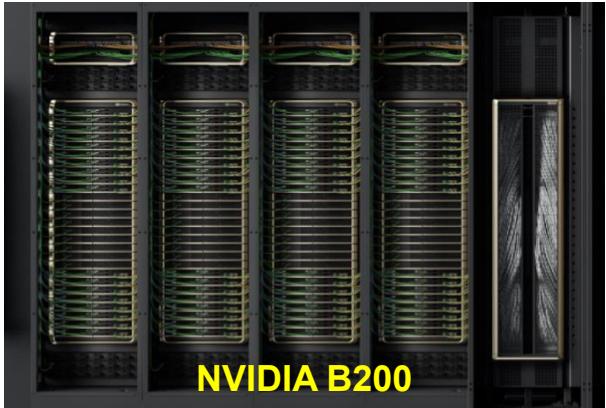
Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

Argonne 
NATIONAL LABORATORY

LARGE COLLABORATION

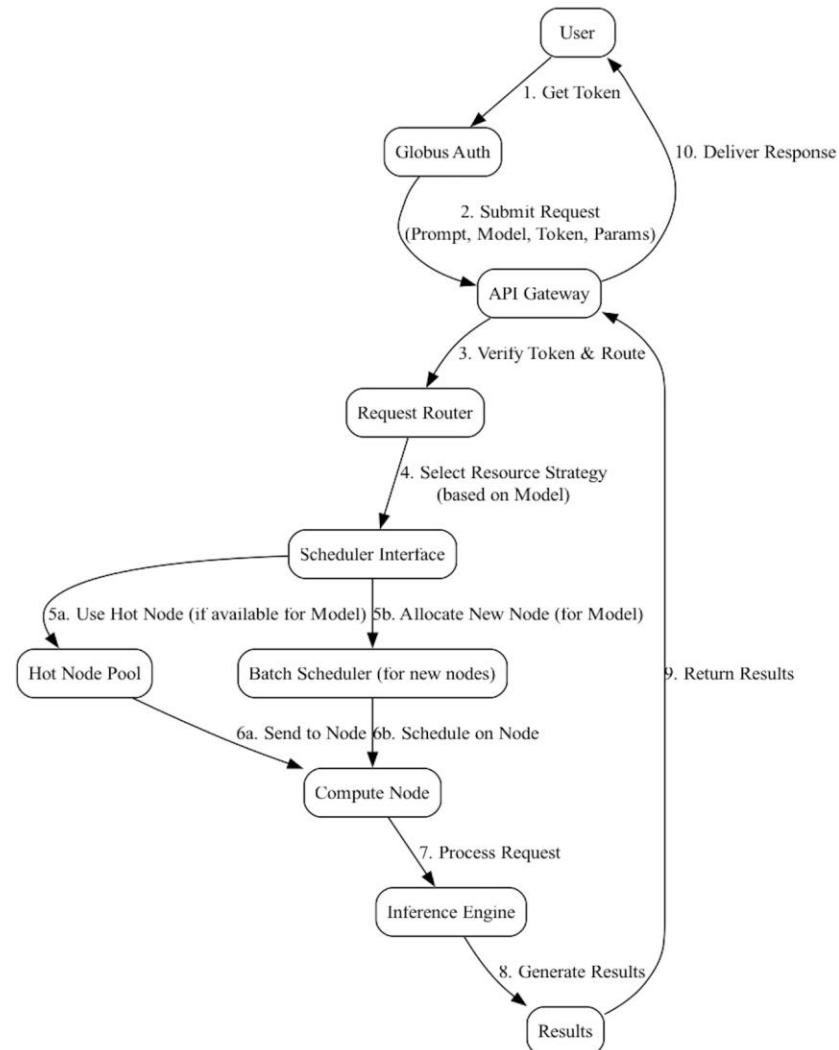
- Ryan Chard, Nick Saint, Tom Uram, Rajeev Thakur, Ken Raffenetti, Le Chen, Yanfei Guo, Krishna Chetty, Murali Emani, Khalid Hossain, Nathan Nichols, Rachana Ananthakrishnan, Anthony Avarca, Bill Allcock, Tommie Jackson, Ian Foster, Mike Papka, Rick Stevens, ALCF and CELS Operations, Globus, Aurora-GPT, and many more.
- Joint collaboration with Globus

ALCF IS DEPLOYING DIVERSE INFERENCE SYSTEMS FOR SCIENCE



WORKFLOW OVERVIEW

- User generates an access token (API key)
- User submits LLM request to Gateway API
- API validates user and request arguments
- API routes request to remote HPC resources
- The service submits a job to the local scheduler to load the model (or reuses already-acquired compute nodes)
- The service executes the LLM request on the behalf of the user
- Result is returned to the Gateway API
- Result is returned to the user



PRESENTATION OVERVIEW

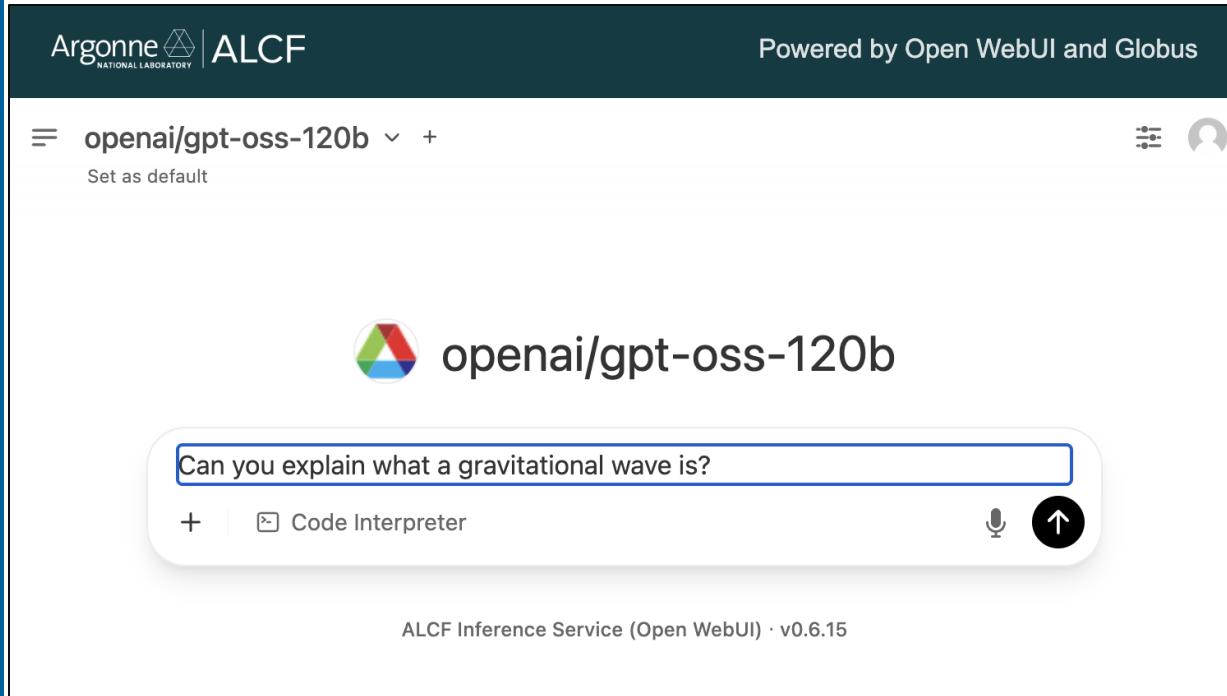
- **Overview of the User Interface**
 - Entry points to our inference service
 - Motivation behind our development
- **ALCF Inference Service Architecture**
 - Overview of the system components
 - Authentication and authorization
 - Orchestration and configuration
 - Available models
- **Capabilities and Features**
 - Latency, scaling, federated endpoints
 - Monitoring, production-ready with containers
- **Usage and Examples**
 - How to use the API (Python, cURL, agentic workflows)
 - How to use the web interface
 - Science use cases

PRESENTATION OVERVIEW

- **Overview of the User Interface**
 - Entry points to our inference service
 - Motivation behind our development
- **ALCF Inference Service Architecture**
 - Overview of the system components
 - Authentication and authorization
 - Orchestration and configuration
 - Available models
- **Capabilities and Features**
 - Latency, scaling, federated endpoints
 - Monitoring, production-ready with containers
- **Usage and Examples**
 - How to use the API (Python, cURL, agentic workflows)
 - How to use the web interface
 - Science use cases

INTERFACING WITH THE INFERENCE SERVICE

Open WebUI interface
<https://inference.alcf.anl.gov>



The screenshot shows the ALCF Inference Service interface. At the top, there's a dark header bar with the Argonne National Laboratory logo and the text "ALCF". To the right of the header, it says "Powered by Open WebUI and Globus". Below the header, the main content area has a title "openai/gpt-oss-120b" with a dropdown arrow and a "+" sign. Underneath this, there's a "Set as default" link. On the right side of the main area, there are three icons: a person icon, a gear icon, and an upward-pointing arrow icon. The main content area itself has a white background. It features a circular icon with red, green, and blue segments followed by the text "openai/gpt-oss-120b". Below this, there's a text input field containing the question "Can you explain what a gravitational wave is?". To the left of the input field is a "+" button and a "Code Interpreter" checkbox. To the right of the input field are a microphone icon and an upward-pointing arrow icon. At the bottom of the main content area, it says "ALCF Inference Service (Open WebUI) · v0.6.15".

INTERFACING WITH THE INFERENCE SERVICE

Open WebUI interface
<https://inference.alcf.anl.gov>

Argonne | ALCF

Powered by Open WebUI and Globus

openai/gpt-oss-120b

Set as default

openai/gpt-oss-120b

Can you explain what a gravitational wave is?

+ Code Interpreter

ALCF Inference Service (Open WebUI) · v0.6.15

Argonne | ALCF

Powered by Open WebUI and Globus

openai/gpt-oss-120b

2. How Gravitational Waves Are Produced

Gravitational waves are generated whenever **mass-energy accelerates asymmetrically**. The strongest astrophysical sources are:

SOURCE	TYPICAL FREQUENCY (HZ)	TYPICAL STRAIN (H) AT EARTH
Binary Black Hole (BH-BH) Merger	10-500	$10^{-21} - 10^{-22}$
Binary Neutron Star (NS-NS) Merger	10-2000	$10^{-21} - 10^{-23}$
Supernova Core Collapse	$\sim 100 - 1000$	$10^{-23} - 10^{-24}$
Rapidly Rotating Neutron Stars (mountains)	$\sim 10 - 1000$	$10^{-26} - 10^{-28}$
Stochastic Background (early universe)	$10^{-9} - 10^3$	Extremely tiny, model-dependent

Key points:

Quadrupole Moment: The dominant contribution to gravitational radiation comes from the second time derivative of the mass quadrupole moment Q_{ij} . A changing monopole (total mass) or dipole (center-of-mass motion) does **not** radiate in GR.

Energy Loss: As a system radiates, it loses orbital energy, causing binary components to spiral inward (the "chirp" observed by detectors).

3. What a Gravitational Wave Does to Space

When a GW passes, it **stretches** space in one direction while **compressing** it in the perpendicular direction, then swaps the roles as the wave oscillates. For a wave traveling along the z -axis, the metric perturbation in TT (transverse-traceless) gauge looks like:

$$ds^2 = -c^2 dt^2 + [1 + h_+(t - z/c)] dx^2 + [1 - h_+(t - z/c)] dy^2 + dz^2,$$

+ Code Interpreter

INTERFACING WITH THE INFERENCE SERVICE

OpenAI API (including batch)

<https://docs.alcf.anl.gov/services/inference-endpoints>

cURL Python (OpenAI SDK)

```
#!/bin/bash

# Get your access token
access_token=$(python inference_auth_token.py get_access_token)

curl -X POST "https://inference-api.alcf.anl.gov/resource_server/sophia/vllm/v1/chat/completions" \
-H "Authorization: Bearer ${access_token}" \
-H "Content-Type: application/json" \
-d '{
    "model": "meta-llama/Meta-Llama-3.1-8B-Instruct",
    "messages": [{"role": "user", "content": "Explain quantum computing in simple terms."}]
}'
```

API Usage Examples

Querying Endpoint Status

 [Querying Endpoint Status](#) >

Chat Completions

 [Chat Completions](#) >

INTERFACING WITH THE INFERENCE SERVICE

OpenAI API (including batch)

<https://docs.alcf.anl.gov/services/inference-endpoints>

cURL Python (OpenAI SDK)

```
#!/bin/bash

# Get your access token
access_token=$(python inference_auth_token.py get_access_token)

curl -X POST "https://inference-api.alcf.anl.gov/resource_server/sophia"
  -H "Authorization: Bearer ${access_token}" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "meta-llama/Meta-Llama-3.1-8B-Instruct",
    "messages": [{"role": "user", "content": "Explain quantum co"}'
```

```
{
  "id": "chatcmpl-68de443dde8b46659b4c34",
  "object": "chat.completion",
  "created": 1755114580,
  "model": "meta-llama/Meta-Llama-3.1-8B",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Quantum computing is a new way of processing information that's different from the way regular computers work. Here's a simplified explanation:\n\nRegular Computers: Regular computers use \"bits\" to store and process information. Bits are like light switches that can be either ON (1) or OFF (0). When you combine these bits, you get numbers, letters, and other data.\n\nQuantum Computers: Quantum computers use \"qubits\" (quantum bits) to store and process information. Qubits are special because they can be both ON and OFF at the same time, which is called a \"superposition.\" This means a qubit can process multiple possibilities simultaneously, making it much faster than regular computers for certain tasks.\n\nAnother Key Concept: Entanglement\nQubits can also be \"entangled,\" which means that when something happens to one qubit, it instantly affects the other qubits, no matter how far apart they are. This allows quantum computers to perform calculations on multiple qubits simultaneously, making them incredibly powerful.\n\nHow Quantum Computing Works\nImagine you have a combination lock with 10 numbers. A regular computer would try each number one by one, taking a long time to find the correct combination. A quantum computer, on the other hand, can try all 10 numbers simultaneously, thanks to the power of qubits and entanglement. This makes quantum computing incredibly fast for certain tasks, such as:\n1. **Cryptography:** Breaking complex codes and encryption methods.\n2. **Optimization:** Finding the best solution for complex problems, like logistics and supply chain management.\n3. **Simulation:** Simulating complex systems, like weather patterns and molecular interactions.\n\nChallenges and Limitations\nQuantum computing is still a developing field, and there are many challenges to overcome, such as:\n1. **Error correction:** Qubits are prone to errors, which can affect the accuracy of calculations.\n2. **Scalability:** Currently, quantum computers are small and can only perform a limited number of calculations.\n3. **Noise:** Quantum computers are sensitive to external noise, which can disrupt calculations.\n\nConclusion\nQuantum computing is a revolutionary technology that has the potential to solve complex problems that are currently unsolvable or take too long to solve with regular computers. While it's still in its early stages, researchers and companies are working to overcome the challenges and limitations, and we can expect to see significant advancements in the coming years."}
```

```
"usage": {
  "prompt_tokens": 43,
  "total_tokens": 436,
  "completion_tokens": 393,
  "prompt_tokens_details": null
},
"prompt_logprobs": null,
"kv_transfer_params": null,
"response_time": 3.179178237915039,
"throughput_tokens_per_second": 137.14235798428732
```

API Usage Examples

Querying Endpoint Status

 [Querying Endpoint Status](#)

Chat Completions

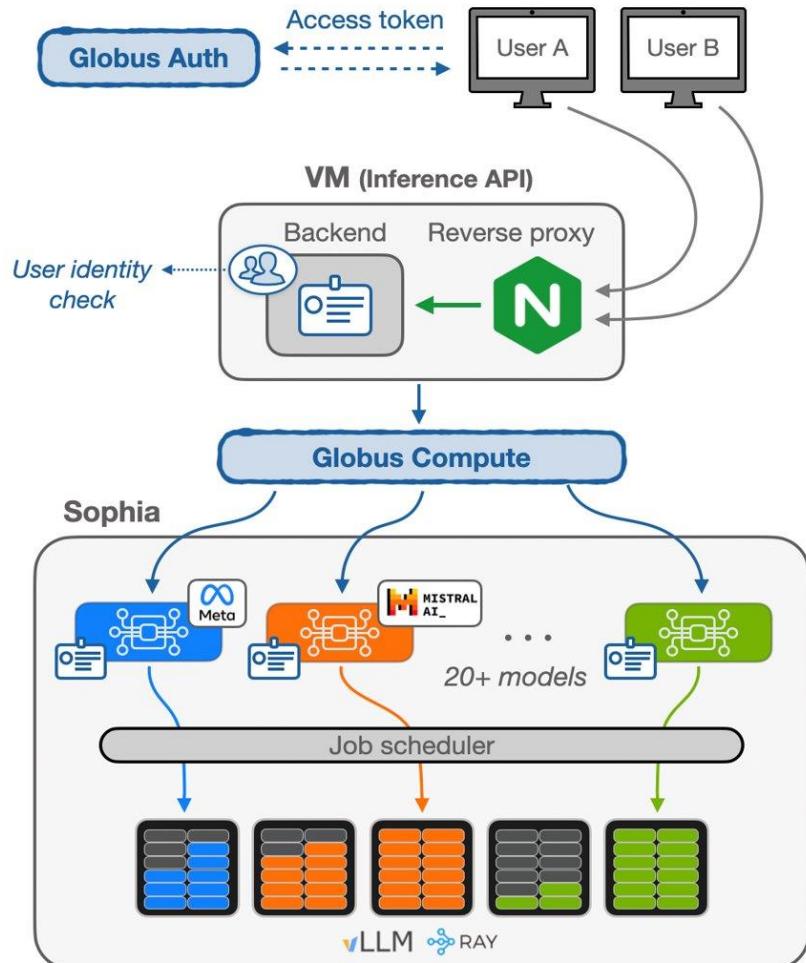
 [Chat Completions](#)

PRESENTATION OVERVIEW

- **Overview of the User Interface**
 - Entry points to our inference service
 - Motivation behind our development
- **ALCF Inference Service Architecture**
 - Overview of the system components
 - Authentication and authorization
 - Orchestration and configuration
 - Available models
- **Capabilities and Features**
 - Latency, scaling, federated endpoints
 - Monitoring, production-ready with containers
- **Usage and Examples**
 - How to use the API (Python, cURL, agentic workflows)
 - How to use the web interface
 - Science use cases

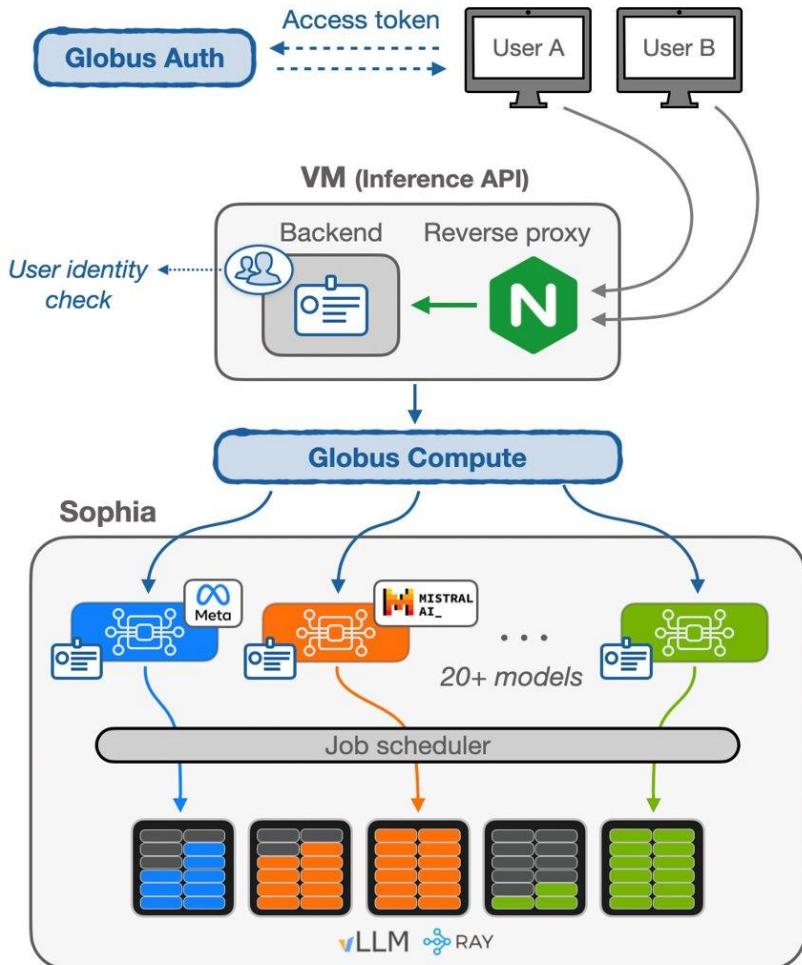
SYSTEM OVERVIEW

- Inference Service leverages ALCF computing resources to serve requests to a growing set of models
- Authentication via Globus Auth and orchestration using Globus Compute
- Combination of “in-memory/active” models and on-demand schedulable models
- Usage metrics are curated to understand and improve the service



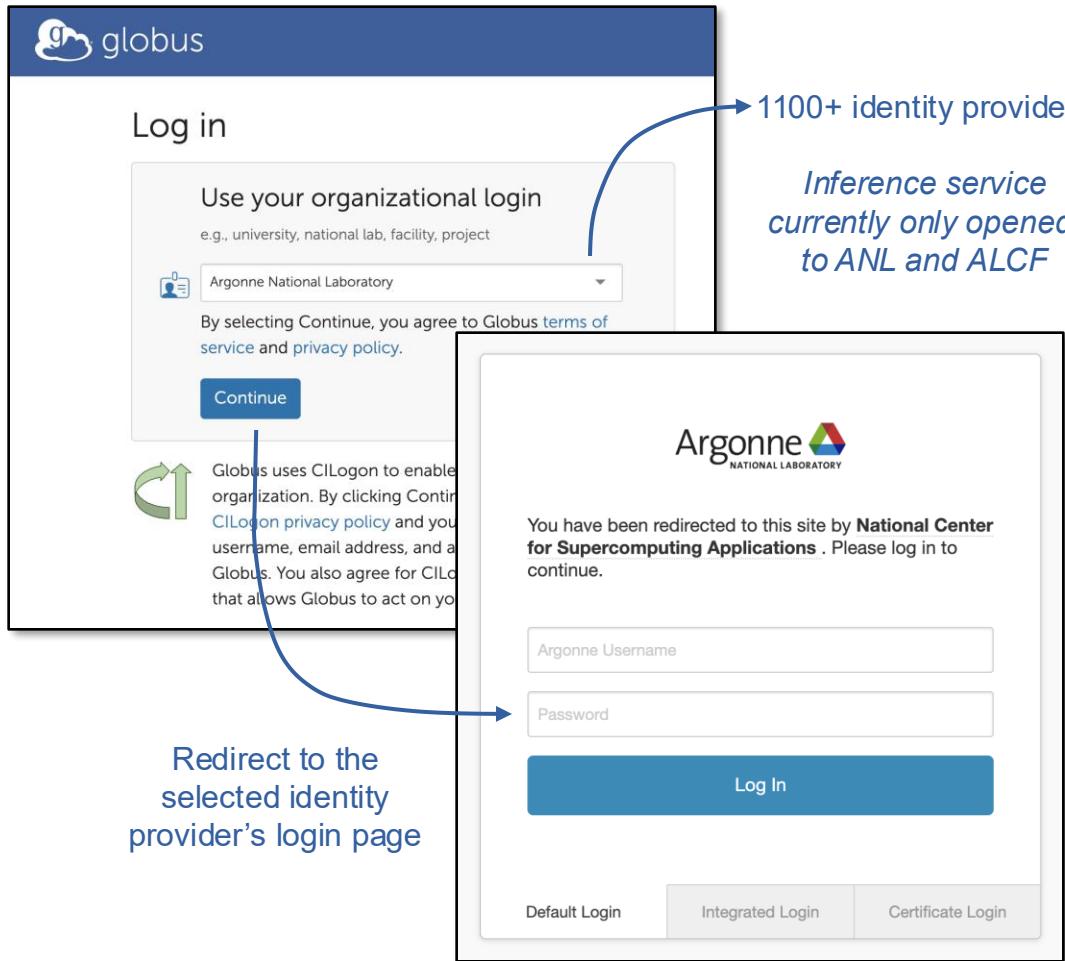
SYSTEM COMPONENTS

- **Globus Auth:** Enterprise-grade authentication and authorization service (OAuth2/OpenID)
- **API Gateway:** Django-Ninja async, OpenAI-compliant API handling, authorization and request routing, Postgres DB, monitoring
- **Globus Compute:** Orchestration and remote execution framework on HPC clusters
- **Compute Resources:** Compute nodes with GPUs on the ALCF Sophia cluster (more coming...)
- **Inference Backend:** High-performance inference servers (e.g., vLLM) for model serving; model weights downloaded and stored on HPC cluster



GLOBUS AUTH

- Authentication and authorization platform (OAuth2/OpenID compliant)
- Federated identity provider integrating with different institutions worldwide



GLOBUS AUTH

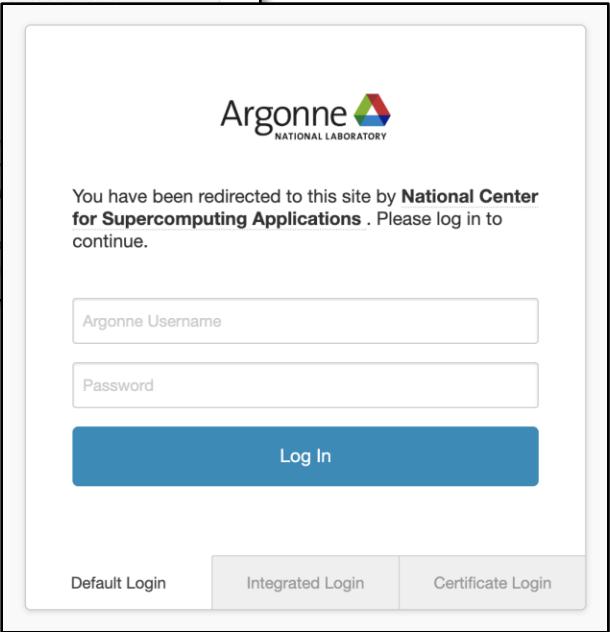
- Authentication and authorization platform (OAuth2/OpenID compliant)
- Federated identity provider integrating with different institutions worldwide
- From a user's perspective:
 - Globus Auth generates a token
 - The token is passed to our inference service as an API key

```
client = OpenAI(  
    api_key=access_token,  
    base_url="https://inference-api.alcf.anl.gov/resource_server/sophia/vllm/v1"  
)  
  
response = client.chat.completions.create(  
    model="meta-llama/Meta-Llama-3.1-8B-Instruct",  
    messages=[{"role": "user", "content": "What are the symptoms of diabetes?"}]  
)
```



The screenshot shows the Globus Auth login page. It features a dropdown menu for selecting an organizational login, currently set to "Argonne National Laboratory". A note below the dropdown states: "By selecting Continue, you agree to Globus terms of service and privacy policy." A blue arrow points from the text "1100+ identity providers" to the top right of the dropdown area.

Inference service currently only opened to ANL and ALCF

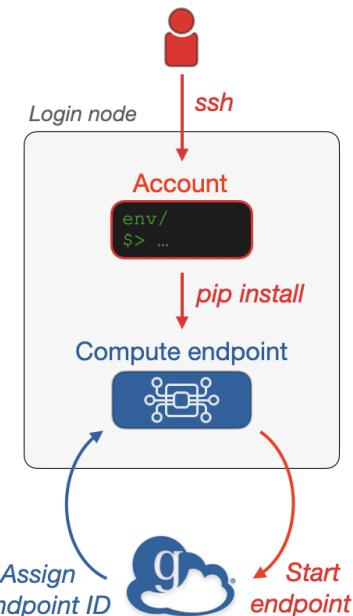


The screenshot shows the Argonne National Laboratory login page. It displays the Argonne logo and a message: "You have been redirected to this site by National Center for Supercomputing Applications. Please log in to continue." Below this are fields for "Argonne Username" and "Password", and a "Log In" button. At the bottom, there are links for "Default Login", "Integrated Login", and "Certificate Login".

GLOBUS COMPUTE

Globus Compute can **trigger remote analysis** on HPC systems from anywhere in the world. **Endpoints** deployed on login nodes submit jobs to the scheduler to execute Python **functions**.

Install endpoint



Register function

```
# Create Globus Compute client
from globus_compute_sdk import Client
gcc = Client()
```

The function can do whatever you want, including writing data on the filesystem or call more complex codes.

```
# Define your analysis function
def my_analysis(arguments):

    # Import necessary modules
    import numpy as np
    import scipy

    # Do some analysis using local codes
    # ...

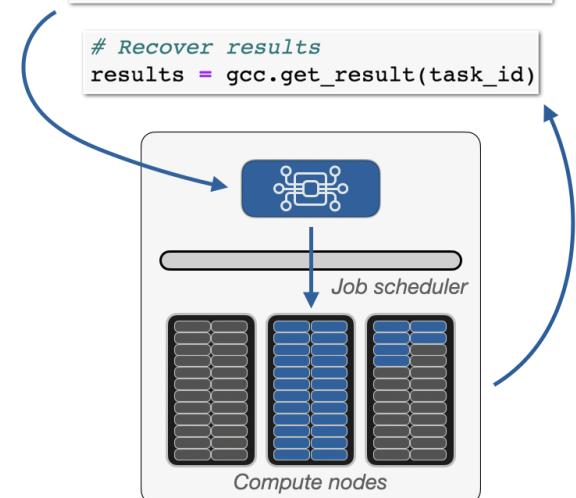
    # Return the computation results
    return ...
```

```
# Register your function
function_id = gcc.register_function(my_analysis)
```

Run analysis

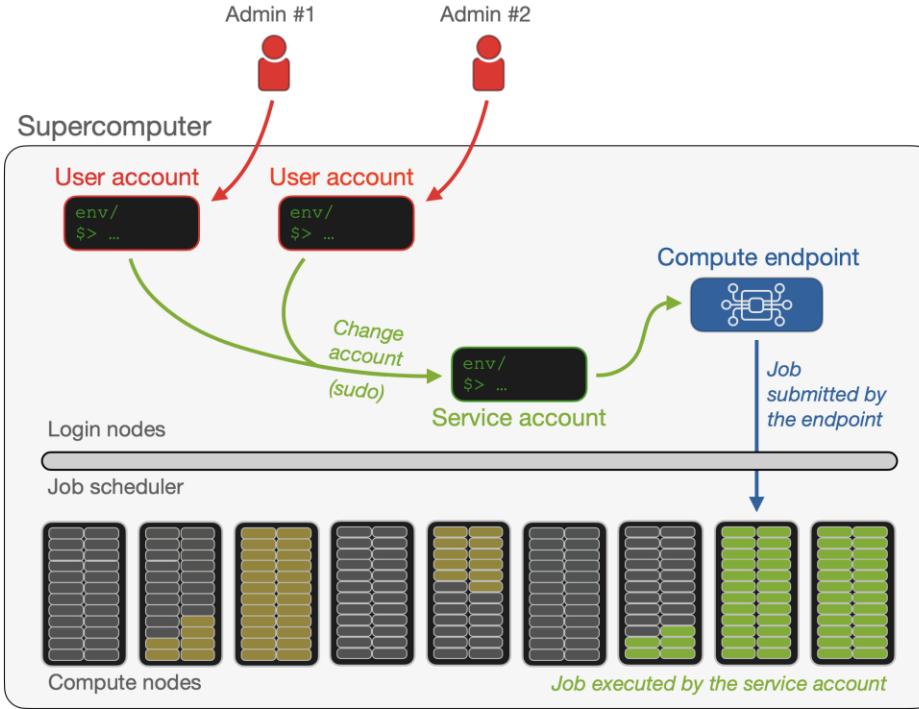
```
# Submit a function to an endpoint
task_id = gcc.run(
    "my_arguments"
    endpoint_id=endpoint_id,
    function_id=function_id)
```

```
# Recover results
results = gcc.get_result(task_id)
```



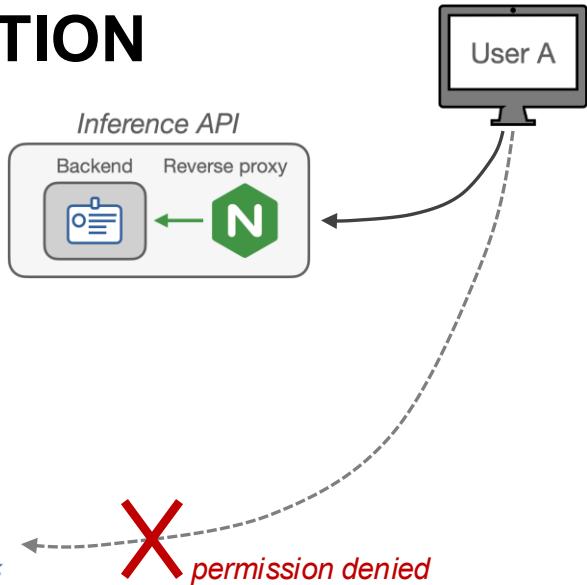
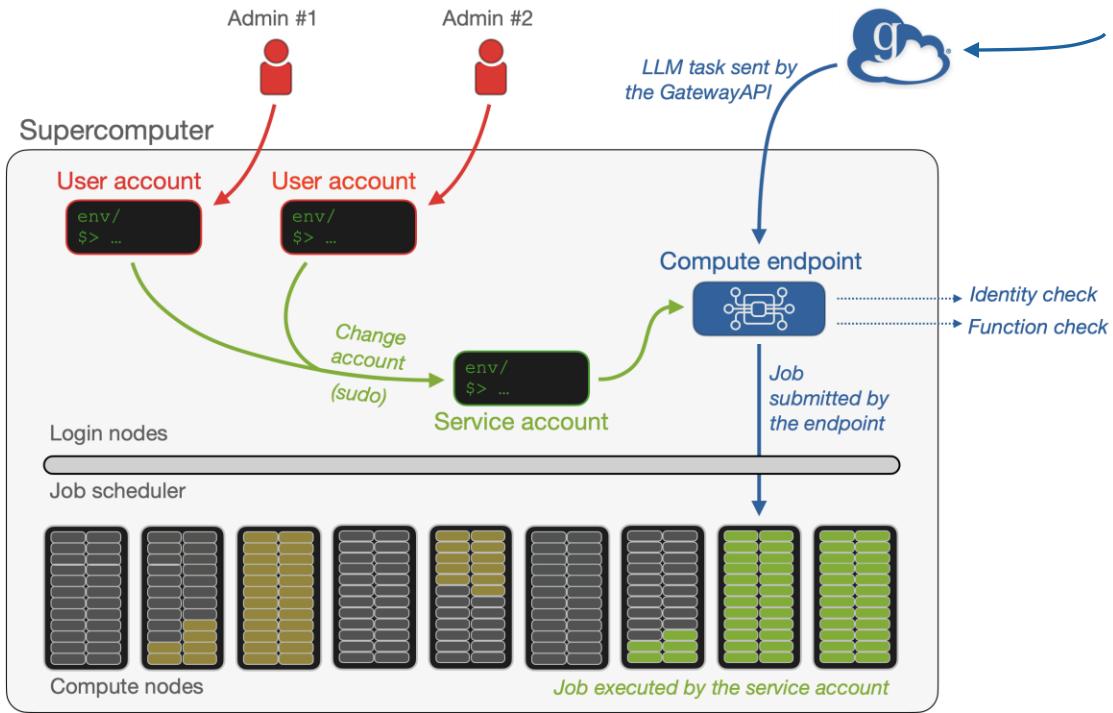
INFERENCE ENDPOINT CONFIGURATION

The **system administrators** deploy and configure the **compute endpoints** from an **ALCF service account**



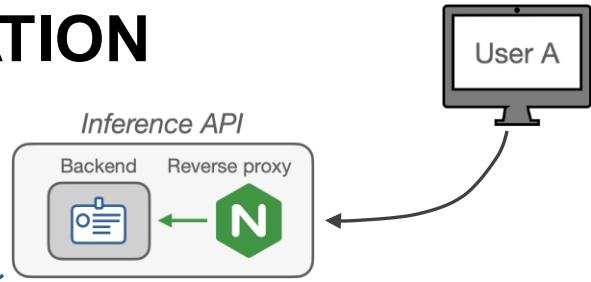
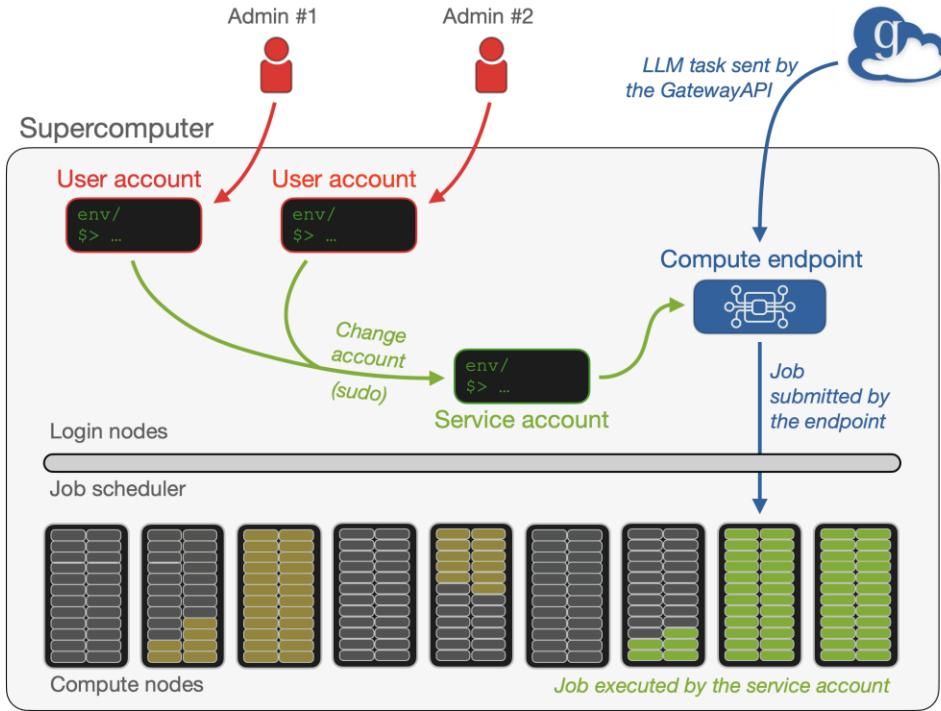
INFERENCE ENDPOINT CONFIGURATION

The **system administrators** deploy and configure the **compute endpoints** from an **ALCF service account**



INFERENCE ENDPOINT CONFIGURATION

The **system administrators** deploy and configure the **compute endpoints** from an **ALCF service account**



Model initialization

- Load targeted model(s) into memory
- Fine-tune and start the vLLM server
- Log configuration into local file
- Keep the model hot (for 2 to 24 hours)

Task execution

- Submit task to internal vLLM server
- Log activity into local file
- Return the inference result

AVAILABLE MODELS (30 TOTAL)

B - Batch enabled
T - Tool calling enabled
R - Reasoning enabled

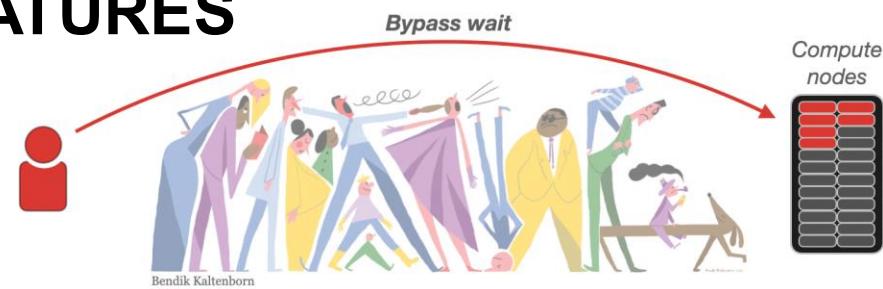
Family	Models
Qwen	Qwen2.5-14B-Instruct ^{BT} , Qwen2.5-7B-Instruct ^{BT} , QwQ-32B ^{BR} , Qwen3-235B-A22B ^{RT} , Qwen3-32B ^{BR}
Meta Llama	Meta-Llama-3-70B-Instruct ^B , Meta-Llama-3-8B-Instruct ^B , Meta-Llama-3.1-70B-Instruct ^{BT} , Meta-Llama-3.1-8B-Instruct ^{BT} , Meta-Llama-3.1-405B-Instruct ^{BT} , Llama-3.3-70B-Instruct ^{BT} , Llama-4-Scout-17B-16E-Instruct ^{BT} , Llama-4-Maverick-17B-128E-Instruct ^T
OpenAI	gpt-oss-20b, gpt-oss-120b
Mistral	Mistral-7B-Instruct-v0.3 ^B , Mistral-Large-Instruct-2407 ^B , Mixtral-8x22B-Instruct-v0.1 ^B
Nemotron	mgoin/Nemotron-4-340B-Instruct-hf
Aurora GPT	AuroraGPT-IT-v4-0125 ^B , AuroraGPT-Tulu3-SFT-0125 ^B , AuroraGPT-DPO-UFB-0225 ^B , AuroraGPT-7B-OI ^B
Allenai	Llama-3.1-Tulu-3-405B
Google	gemma-3-27b-it ^{BT}
Vision (VLM)	Qwen/Qwen2-VL-72B-Instruct ^B , meta-llama/Llama-3.2-90B-Vision-Instruct
Embedding	nvidia/NV-Embed-v2, Salesforce/SFR-Embedding-Mistral ^B , mistralai/Mistral-7B-Instruct-v0.3-embed ^B

PRESENTATION OVERVIEW

- **Overview of the User Interface**
 - Entry points to our inference service
 - Motivation behind our development
- **ALCF Inference Service Architecture**
 - Overview of the system components
 - Authentication and authorization
 - Orchestration and configuration
 - Available models
- **Capabilities and Features**
 - Latency, scaling, federated endpoints
 - Monitoring, production-ready with containers
- **Usage and Examples**
 - How to use the API (Python, cURL, agentic workflows)
 - How to use the web interface
 - Science use cases

KEY CAPABILITIES AND FEATURES

- **Dedicated Compute Resources:** Selected LLMs persistently served on dedicated nodes. This bypasses HPC queues and “cold starts”.

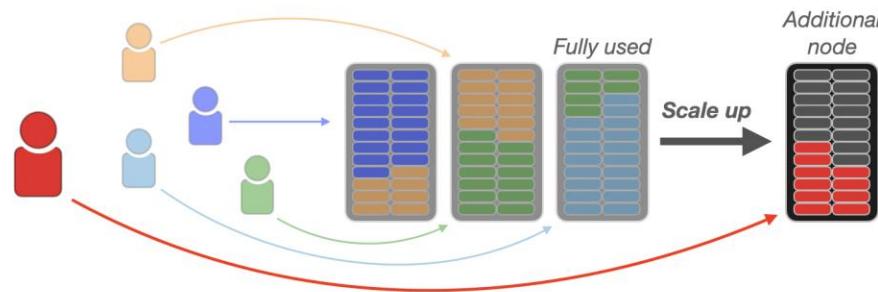


KEY CAPABILITIES AND FEATURES

- **Dedicated Compute Resources:** Selected LLMs persistently served on dedicated nodes. This bypasses HPC queues and “cold starts”.



- **Auto-Scaling and Hot Nodes:** New nodes can dynamically be acquired to accommodate higher traffic. Cold models can be dynamically be loaded and kept hot for 24 hours.

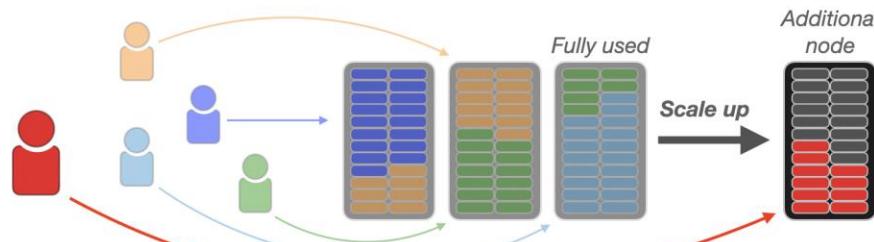


KEY CAPABILITIES AND FEATURES

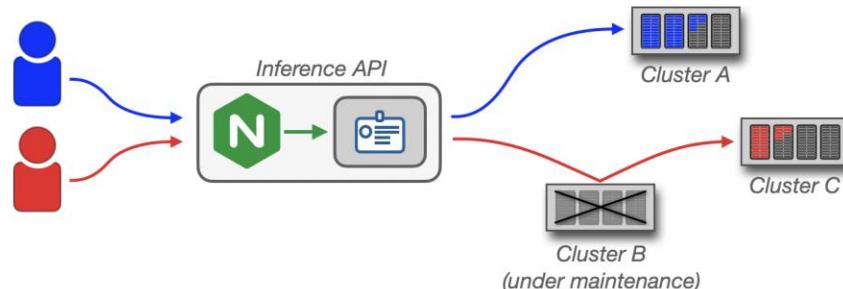
- **Dedicated Compute Resources:** Selected LLMs persistently served on dedicated nodes. This bypasses HPC queues and “cold starts”.



- **Auto-Scaling and Hot Nodes:** New nodes can dynamically be acquired to accommodate higher traffic. Cold models can be dynamically be loaded and kept hot for 24 hours.

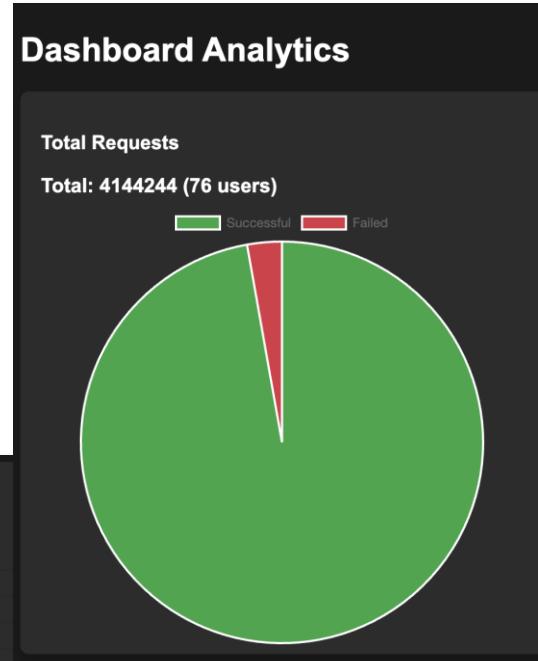
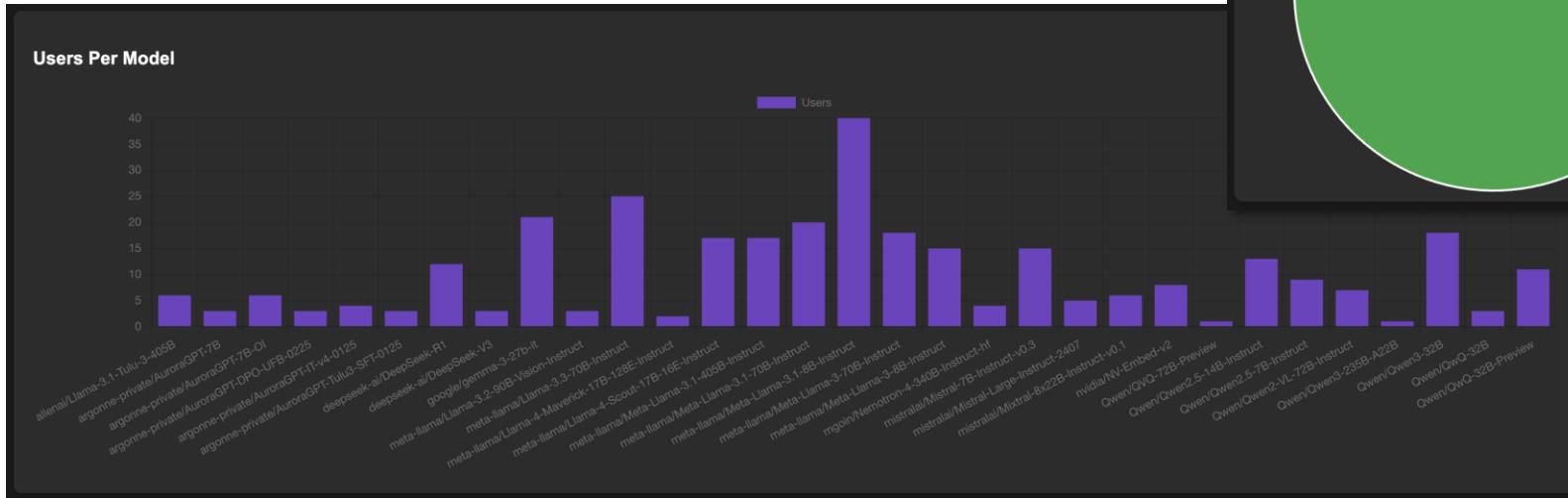


- **Multi-Backend Integration:** Our API can seamlessly route requests to diverse remote hardware, including SambaNova SN40 and Sophia inference clusters.



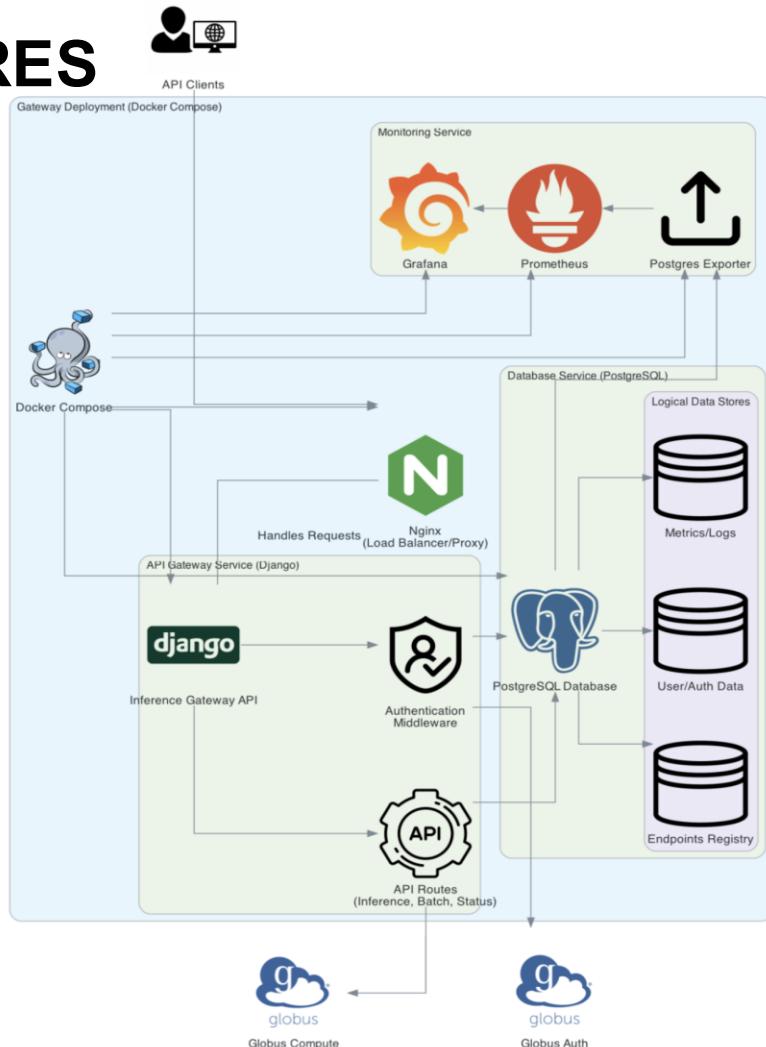
KEY CAPABILITIES AND FEATURES

- **Dashboard Monitoring:** A dashboard is available to system administrators and provides various metrics such as recent activities, number of requests and users, token throughput, and latency.
- **Current Status:** Over 8.7M requests, over 10 billion tokens generated, can generate ~3,500 tokens per second on a Sophia compute node.



KEY CAPABILITIES AND FEATURES

- **Production Ready:** Container deployment
 - Django application
 - Gunicorn/Uvicorn + Nginx
 - PostgreSQL
 - Redis cache management
 - Comprehensive logging
 - Monitoring with Grafana and Prometheus
 - Globus integration
- **Benchmarking:** Built-in load testing tools
 - Request throughput
 - Token throughput
 - Latency



PRESENTATION OVERVIEW

- **Overview of the User Interface**
 - Entry points to our inference service
 - Motivation behind our development
- **ALCF Inference Service Architecture**
 - Overview of the system components
 - Authentication and authorization
 - Orchestration and configuration
 - Available models
- **Capabilities and Features**
 - Latency, scaling, federated endpoints
 - Monitoring, production-ready with containers
- **Usage and Examples**
 - How to use the API (Python, cURL, agentic workflows)
 - How to use the web interface
 - Science use cases

USAGE: AUTHENTICATION

- All API requests require a Globus access token using **ANL or ALCF credentials**.
- Download the helper script: wget
`https://raw.githubusercontent.com/argonne-lcf/inference-endpoints/main/inference_auth_token.py`
- Requires: pip install globus-sdk
- **Token Validity:** 48 hours, but get_access_token auto-refreshes for up to 7 days
- **Troubleshooting:** If auth errors occur, visit <https://app.globus.org/logout> and re-authenticate with --force

USAGE: AUTHENTICATION (CODE)

```
# Authenticate (first time or when token expires)
python inference_auth_token.py authenticate --force

# Get a valid access token
export MY_TOKEN=$(python inference_auth_token.py get_access_token)
```

CHECKING SYSTEM STATUS

- **Check running models and jobs:** See which models are currently live and available
- **List all available endpoints:** Get information about all configured endpoints
- These commands help you understand what's currently available before making requests

CHECKING RUNNING JOBS

```
# Check running jobs and models on the cluster
curl -X GET "https://inference-api.alcf.anl.gov/resource_server/sophia/jobs" \
-H "Authorization: Bearer $MY_TOKEN"
```

```
{ "running": [ { "Models": "meta-llama/Llama-4-Scout-17B-16E-Instruct",
  "Framework": "vllm", "Cluster": "sophia", "Model Status": "running", "Job ID": "80514.sophia-pbs-01", "Job State": "R", "Host Name": "sophia-gpu-15/0*256", "Job Comments": "Job run at Mon Aug 11 at 20:52 on (sophia-gpu-15:ncpus=256:ngpus=8:mem=1006632960kb)", "Nodes Reserved": "1", "Walltime": "67:43:56" } , "queued": [], "others": [], "private-batch-running": [], "private-batch-queued": [], "cluster_status": { "free_nodes": 4 } } }
```

LIST ALL AVAILABLE ENDPOINTS

```
# Check running jobs and models on the cluster
curl -X GET "https://inference-api.alcf.anl.gov/resource_server/list-
endpoints" -H "Authorization: Bearer $MY_TOKEN"
```

```
{"clusters": {"sophia": {"base_url": "/resource_server/sophia", "frameworks": {"vllm": {"models": ["Qwen/QwQ-32B"], "endpoints": {"chat": "/vllm/v1/chat/completions/", "embedding": "/vllm/v1/embeddings/"}}}, "infinity": {"models": ["nvidia/NV-Embed-v2"]}}}, "polaris": {"base_url": "/resource_server/polaris", "frameworks": {"vllm": {"models": ["meta-llama/Meta-Llama-3-70B-Instruct"]}, "endpoints": {"chat": "/vllm/v1/chat/completions/", "completion": "/vllm/v1/completions/", "embedding": "/vllm/v1/embeddings/"}}}}
```

EXAMPLE: CHAT COMPLETION WITH CURL

- Use the `/v1/chat/completions` endpoint for interactive queries.
- Provide the model name and a series of messages.
- This is ideal for testing and simple command-line interactions.



U.S. DEPARTMENT
of ENERGY

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

EXAMPLE: CHAT COMPLETION WITH CURL (CODE)

```
curl -X POST https://inference-
api.alcf.anl.gov/resource_server/v1/chat/completions
\\
    -H "Authorization: Bearer $MY_TOKEN" \\
    -H "Content-Type: application/json" \\
    -d '{
        "model": "meta-llama/Meta-Llama-3.1-8B-
Instruct",
        "messages": [
            {"role": "user", "content": "Explain Globus
Compute in simple terms."}
        ],
        "max_tokens": 150
    }'
```

EXAMPLE: CHAT COMPLETION WITH PYTHON

- Programmatic access is easy with Python's `requests` library.
- This allows for integration into larger applications and workflows.
- Remember to handle the response, which will be in JSON format.

EXAMPLE: CHAT COMPLETION WITH PYTHON (CODE)

```
import requests
from inference_auth_token import get_access_token

access_token = get_access_token()
headers = { 'Authorization': f'Bearer {access_token}' }
url = "https://inference-
api.alcf.anl.gov/resource_server/v1/chat/completions"

payload = {
    "model": "meta-llama/Meta-Llama-3.1-8B-Instruct",
    "messages": [{"role": "user", "content": "Explain Globus
Compute in simple terms."}]
}

response = requests.post(url, headers=headers, json=payload)
print(response.json())
```

EXAMPLE: USING THE OPENAI SDK

- The endpoints are compatible with the official OpenAI Python library.
- Set the `api_key` to your access token and the `base_url` to the ALCF endpoint.
- This provides a familiar and powerful interface for interacting with the models.

EXAMPLE: USING THE OPENAI SDK (CODE)

```
from openai import OpenAI
from inference_auth_token import get_access_token

client = OpenAI(
    api_key=get_access_token(),
    base_url="https://inference-
api.alcf.anl.gov/resource_server/sophia/vllm/v1"
)

response = client.chat.completions.create(
    model="meta-llama/Meta-Llama-3.1-8B-Instruct",
    messages=[{"role": "user", "content": "Explain quantum computing"}]
)
print(response)
```

EXAMPLE: VISION LANGUAGE MODEL (VLM)

- Send image data along with text prompts to Vision Language Models.
- The image must be base64 encoded.
- This enables powerful multi-modal analysis of visual data.



U.S. DEPARTMENT
of ENERGY

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

EXAMPLE: VISION LANGUAGE MODEL (VLM) (CODE)

```
from openai import OpenAI
import base64
from inference_auth_token import get_access_token

client = OpenAI(
    api_key=get_access_token(),
    base_url="https://inference-api.alcf.anl.gov/resource_server/sophia/vllm/v1"
)

# Function to encode image to base64
def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode('utf-8')

base64_image = encode_image("scientific_diagram.png")

response = client.chat.completions.create(
    model="Qwen/Qwen2-VL-72B-Instruct",
    messages=[{"role": "user", "content": [
        {"type": "text", "text": "Describe this scientific diagram"},
        {"type": "image_url", "image_url": {"url": f"data:image/png;base64,{base64_image}"}}
    ]}]
)
```

EXAMPLE: EMBEDDING MODELS

- Generate numerical representations (embeddings) of text for tasks like semantic search or clustering.
- Use the `/embeddings` endpoint. Note that some models may use a different framework endpoint (e.g. infinity).

EXAMPLE: EMBEDDING MODELS (CODE)

```
from openai import OpenAI
from inference_auth_token import get_access_token

client = OpenAI(
    api_key=get_access_token(),
    # Note the different base URL for this embedding model framework
    base_url="https://inference-
api.alcf.anl.gov/resource_server/sophia/vllm/v1"
)

embedding = client.embeddings.create(
    model="nvidia/NV-Embed-v2",
    input="The food was delicious and the waiter was friendly.",
    encoding_format="float"
)
print(embedding)
```

EXAMPLE: BUILDING AN AGENT WITH LANGGRAPH

- Combine LLMs with custom tools to create powerful, automated agents.
- LangGraph's `create_react_agent` can build an agent that uses tools to answer questions.
- This example creates a chemistry agent that can look up molecule structures.

EXAMPLE: BUILDING AN AGENT WITH LANGGRAPH (CODE)

```
from langchain_openai import ChatOpenAI
from langchain_core.tools import tool
from langgraph.prebuilt import create_react_agent
import pubchempy

llm = ChatOpenAI( # Initialize Model
    model="meta-llama/Meta-Llama-3.3-70B-Instruct",
    base_url="https://inference-api.alcf.anl.gov/resource_server/sophia/vllm/v1",
    api_key=get_access_token()
)

@tool # Define Tool
def molecule_name_to_smiles(name: str) -> str:
    """Convert a molecule name to its SMILES format."""
    return pubchempy.get_compounds(str(name), "name")[0].canonical_smiles

chem_graph = create_react_agent(llm, tools=[molecule_name_to_smiles]) # Create Agent
inputs = {"messages": [{"user": "What is the SMILES string for Carbon Dioxide?"}]}

for s in chem_graph.stream(inputs, stream_mode="values"): # Call Local Tool + Remote Model
    print(s["messages"][-1])
```

USAGE: BATCH PROCESSING

- For large-scale inference on many prompts.
- Submit a JSON Lines (`.jsonl`) file containing your requests.
- The system processes these requests asynchronously as a single job.
- This is highly efficient for large datasets and avoids rate-limiting issues.

EXAMPLE: CREATING A BATCH JOB

- Use the `/v1/batches` endpoint to submit a batch job.
- Specify the model and the path to your input file on a shared filesystem.
- The file path must be accessible by the remote compute endpoint.

BATCH JOB INPUT FORMAT (JSONL EXAMPLE)

```
{"custom_id": "req-1", "method": "POST", "url": "/v1/chat/completions",  
"body": {"model": "meta-llama/Meta-Llama-3.1-8B-Instruct", "messages":  
[{"role": "user", "content": "Analyze this protein sequence:  
MKTVRQERLK..."}], "max_tokens": 500}}  
{"custom_id": "req-2", "method": "POST", "url": "/v1/chat/completions",  
"body": {"model": "meta-llama/Meta-Llama-3.1-8B-Instruct", "messages":  
[{"role": "user", "content": "What are the implications of climate change on  
Arctic ecosystems?"}], "max_tokens": 300}}  
{"custom_id": "req-3", "method": "POST", "url": "/v1/chat/completions",  
"body": {"model": "meta-llama/Meta-Llama-3.1-8B-Instruct", "messages":  
[{"role": "user", "content": "Explain quantum entanglement in simple  
terms."}], "max_tokens": 200}}
```

EXAMPLE: CREATING A BATCH JOB (CODE)

```
# The base URL points to the batch processing endpoint
base_url="https://inference-api.alcf.anl.gov/resource_server/v1/batches"

# Submit a batch job with the model and input file
curl -X POST "$base_url" \\
      -H "Authorization: Bearer $MY_TOKEN" \\
      -H "Content-Type: application/json" \\
      -d '{
            "model": "meta-llama/Meta-Llama-3.1-8B-Instruct",
            "input_file": "/path/on/shared/storage/my_prompts.jsonl"
          }'
```

EXAMPLE: CHECKING BATCH STATUS

- Monitor the progress of your submitted job using its batch_id.
- The status can be pending, running, completed, or failed.
- This allows you to track long-running jobs without keeping a connection open.

EXAMPLE: CHECKING BATCH STATUS (CODE)

```
# Replace with the actual ID of your submitted batch job
batch_id="your-batch-id"

# Query the status of a specific batch job
curl -X GET "https://inference-
api.alcf.anl.gov/resource_server/v1/batches/${batch_id}" \\
      -H "Authorization: Bearer $MY_TOKEN"
```

EXAMPLE: RETRIEVING BATCH STATUS

- Retrieve the results from submitted job using its `batch_id`.
- You will be provided a `results_file`, `progress_file` and `metrics`

EXAMPLE: RETRIEVING BATCH RESULT(CODE)

```
# Replace with the actual ID of your submitted batch job
batch_id="your-batch-id"

# Return the results of a specific batch job
curl -X GET "https://inference-
api.alcf.anl.gov/resource_server/v1/batches/${batch_id}/result" \\
-H "Authorization: Bearer $MY_TOKEN"
```

INTERACTIVE CHAT UI

URL: <https://inference.alcf.anl.gov/>

The screenshot shows the Argonne ALCF Interactive Chat UI. On the left is a sidebar with various project links: Benchmarking Django REST API, Rubin LSST Project Overview, Testing Bot Response, Previous 30 days, Sun Facts and Information, FastAPI Error Handling, Locust Load Testing, and two entries under the June section: UFW Port 80 Disable and AI Assistant Introduction. The main area shows a search dropdown for "meta-llama/Llama-3.3-70B-Instruct". The dropdown lists several models: google/gemma-3-27b-it (selected), meta-llama/Llama-3.3-70B-Instruct, meta-llama/Llama-4-Scout-17B-16E-Instruct, meta-llama/Meta-Llama-3.1-405B-Instruct, Qwen/Qwen3-32B, and a Temporary Chat option. Below the dropdown is a "Code Interpreter" button. To the right, a modal window titled "Llama-3.3-70B-Instruct" displays two examples: "Programming Example" (Measure the time it takes to sort a list of 1 million random integers using Python's built-in sort function) and "Physics Example" (Derive the Ginzburg-Landau free energy functional).

Argonne ALCF NATIONAL LABORATORY

Powered by Open WebUI and Globus

New Chat

Search

Chats

Today

Benchmarking Django REST API

Rubin LSST Project Overview

Testing Bot Response

Previous 30 days

Sun Facts and Information

FastAPI Error Handling

Locust Load Testing

June

UFW Port 80 Disable

AI Assistant Introduction

meta-llama/Llama-3.3-70B-Instruct

Search a model

Live Starting Queued Offline All

google/gemma-3-27b-it

meta-llama/Llama-3.3-70B-Instruct

meta-llama/Llama-4-Scout-17B-16E-Instruct

meta-llama/Meta-Llama-3.1-405B-Instruct

Qwen/Qwen3-32B

Temporary Chat

+

Code Interpreter

Llama-3.3-70B-Instruct

↳ Suggested

Programming Example

Measure the time it takes to sort a list of 1 million random integers using Python's built-in sort function.

Physics Example

Derive the Ginzburg-Landau free energy functional.

INTERACTIVE CHAT UI

- Based on WebUI project
- Preserves chat history
- Can compare models directly
- Format equations and codes
- Integrates with Inference API
- Integrates with Globus

Nginx Configuration: If you're using Nginx as a reverse proxy, you must be listening on port 80. Deleting the ufw rule alone won't prevent Nginx from preventing the firewall from forwarding those connections to Nginx. Then edit your Nginx site configuration file (usually in /etc/nginx/sites-available) and add server blocks. Then, reload Nginx: sudo systemctl reload nginx

HTTPS Redirection: It's very common to redirect all HTTP (port 80) traffic to HTTPS (port 443). This is the best practice for security. This is often done *within* your Nginx configuration. If you haven't already set up a redirect, now is the time to do it. Here's an example Nginx configuration snippet to redirect:

```
nginx
1 server {
2   listen 80;
3   server_name yourdomain.com www.yourdomain.com;
4   return 301 https://$host$request_uri;
5 }
```

Powered by Open WebUI and Globus

openai/gpt-oss-120b v +

theory (QED) and in the derivation of the electromagnetic stress-energy tensor.

4. Wave Equations and the Speed of Light

Combine the curl equations and use the identities $\nabla \times (\nabla \times \mathbf{E}) = \nabla(\nabla \cdot \mathbf{E}) - \nabla^2 \mathbf{E}$ together with the constitutive relations (for a source-free, linear, homogeneous medium) to obtain:

$$\nabla^2 \mathbf{E} - \mu\epsilon \frac{\partial^2 \mathbf{E}}{\partial t^2} = 0, \quad \nabla^2 \mathbf{B} - \mu\epsilon \frac{\partial^2 \mathbf{B}}{\partial t^2} = 0.$$

These are vector wave equations with propagation speed

$$c_{\text{med}} = \frac{1}{\sqrt{\mu\epsilon}}.$$

In vacuum ($\mu = \mu_0$, $\epsilon = \epsilon_0$) the speed reduces to the universal constant

$$c = \frac{1}{\sqrt{\mu_0\epsilon_0}} \approx 2.998 \times 10^8 \text{ ms}^{-1},$$

which Maxwell famously identified as the speed of light, confirming the deep unity of optics and electromagnetism.

Streaming enabled

INTERACTIVE CHAT UI

URL: <https://inference.alcf.anl.gov/>

Argonne ALCF | Powered by Open WebUI and Globus

meta-llama/Llama-3.3-70B-Instruct +
google/gemma-3-27b-it -
openai/gpt-oss-120b -

What are the popular tools used to build and serve a REST API?

meta-llama/Llama-3...
Popular Tools for Building and Serving a REST API
Here are some popular tools used to build and serve a REST API:

Frameworks
Express.js (Node.js): A lightweight and flexible framework for building web applications and REST APIs.
Django Rest Framework (Python): A high-level framework for building RESTful APIs in Python.

google/gemma-3...
Okay, let's break down popular tools for building and serving REST APIs. I'll categorize them by language/framework, and also include tools for related aspects like testing, documentation, and API gateways.

I. Languages & Frameworks (The Core Builders)
This is where the bulk of the work happens. The choice often depends on your existing skills, team expertise,

openai/gpt-oss-...
Thought for 1 seconds
Below is a quick-reference guide to the most widely-used tools, frameworks, and platforms for **building** and **serving** a REST-style API. I've grouped them by programming language/ecosystem, then added a few cross-language utilities that help you design, document, secure, and deploy the API.

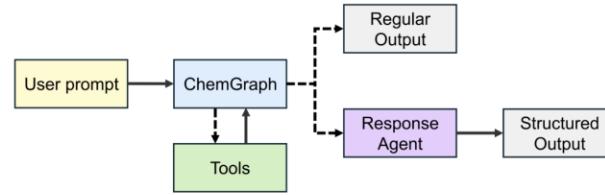
1. JavaScript / Node.js

TOOL **WHAT IT DOES**

+ Code Interpreter

SOME INFERENCE SERVICE USE-CASES

The screenshot shows the ALCF User Support interface. On the left, there's a sidebar with file upload, download, export, and system status options. The main area has a title "ALCF User Support" and a subtitle "Get help with ALCF systems and support tickets". It shows a "User Support Mode" section with a retriever and LLM configuration. Below this is a "Ticket Details" panel for RITM0429668, listing ticket information, subject, and total interactions. A "First Interaction (Preview)" section shows an email template. At the bottom, there's a conversation preview and a message from "You".



Workflow and Cheminformatics

- LangGraph
- ASE

- RDKit
- PubChemPy

Simulation Backends

- Semi-empirical
- Ab initio

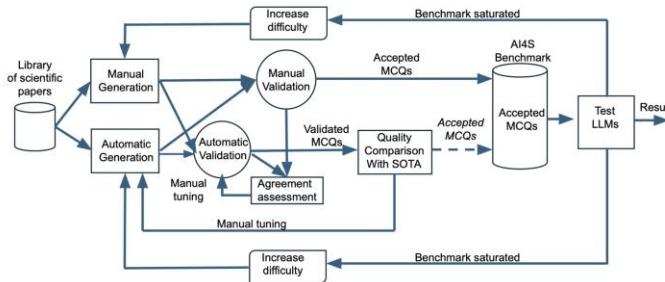
- xTB
- EMT

- NWChem
- ORCA

- MACE
- UMA

ChemGraph – Thang et. al

<https://www.arxiv.org/pdf/2506.06363>



AuroraGPT-EAIRA – Capello et. al (AGIL framework)
<https://arxiv.org/pdf/2502.20309>

Largest usage is currently for generating synthetic data for science

CONCLUSION

ALCF Inference Endpoints democratize LLM access for scientific research by:

- Providing seamless access to 20+ cutting-edge models
- Supporting diverse use cases: chat, vision, embeddings, agents, and batch processing
- Integrating with multiple HPC backends for optimal performance
- Auto scaling based on request workload

Web Interface: <https://inference.alcf.anl.gov/>

Documentation: <https://docs.alcf.anl.gov/services/inference-endpoints/>

Contact: support@alcf.anl.gov

THANK YOU

BACKUP SLIDES

GLOBUS AUTH (FROM AN ADMIN PERSPECTIVE)

Policies define which institutions can use the service

HighAssuranceAccessPolicy

Policy ID	[REDACTED]	
Display Name	HighAssuranceAccessPolicy	
Description	Policy to restrict access to ANL credentials	
High Assurance	Yes	
Authentication Assurance Timeout	7 days	
Domain Constraints Include	alcf.anl.gov anl.gov	

Overview Members Subgroups

2 active 0 pending 0 invited 0 inactive

ACTIVE WAITING INACTIVE

NAME	USERNAME ^	STATUS	ROLE	
Aditya Tanikanti	atanikanti@anl.gov	Active	Member	
Benoit Cote	bcote@alcf.anl.gov	Active	Administrator	

Groups provide role-base access control to private models

Clients operate the service without human intervention

Registered Apps

	InferenceServiceAPI
	InferencePortal
	PublicUserAuthClient
	InferenceGroupMonitor

Invite Others