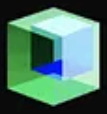
IBM Granite 4.0-Nano

6 min read · 3 days ago

Alain Airom (Ayrom) Follow

Listen Share More

You can use it on your everyday's laptop 🤩

Model	Architecture Type	Model Size	Intended Use
Granite-4.0-H-Small 	Hybrid, Mixture of Experts	32B total / 9B activated	Workhorse model for key enterprise tasks like RAG and agents
Granite-4.0-H-Tiny 	Hybrid, Mixture of Experts	7B total / 1B activated	Designed for low latency and local applications, particularly where the task has long prefill or other scenarios where a MoE model is desired
Granite-4.0-H-Micro 	Hybrid, Dense	3B total	Designed for low latency and local applications, and as a building block to perform key tasks (like function calling) quickly within agentic workflows
Granite-4.0-Micro 	Traditional, Dense	3B total	Alternative option for users when Mamba2 support is not yet optimized (e.g. llama.cpp, PEFT, etc)
Granite-4.0-H-1B 	Dense, Hybrid	1.5B	Ideal models for edge, on-device, and latency-sensitive use cases
Granite-4.0-1B 	Dense, Traditional	1B	Alternative option for users when Mamba2 support is not yet optimized (e.g. llama.cpp, PEFT, etc)
Granite-4.0-H-350M 	Dense, Hybrid	350M	Similar use cases as Granite-4.0-H-1B, but even smaller and cheaper to run
Granite-4.0-350M 	Dense, Traditional	350M	Alternative option for users when Mamba2 support is not yet optimized (e.g. llama.cpp, PEFT, etc)

IBM Granite 4.0 Nano

IBM has introduced the **Granite 4.0 Nano** model family, making a strong commitment to create powerful and useful large language models (LLMs)

specifically optimized for **edge and on-device applications**. These models, which range from approximately 350 million to 1.5 billion parameters, are highlighted for delivering significantly increased capabilities compared to similarly sized models from competitors, as validated across standard benchmarks in areas like General Knowledge, Math, Code, and Safety.

The release comprises four main variants, including models based on a **new, efficient hybrid-SSM architecture** (like the Granite 4.0 H 1B and H 350M) and alternative traditional transformer versions to ensure compatibility with diverse runtimes (such as `llama.cpp`). Crucially, the Granite 4.0 Nano models are built upon the same robust training pipelines and over 15 trillion tokens of data used for the larger Granite 4.0 family.

For broad usability and confidence, all Nano models are released under the **Apache 2.0 license** and come with **IBM's ISO 42001 certification** for responsible model development and governance, allowing users to deploy them with assurance of global standards compliance.

❓ How Try Granite 4 Nano?

These state-of-the-art models are conveniently accessible through two primary channels: the streamlined deployment platform, Ollama, and the comprehensive repository maintained on Hugging Face. Direct access links for both resources are consolidated within the dedicated “Links” section for your convenience. To clearly illustrate the remarkable simplicity and speed of integration, I have adapted and enhanced the foundational code samples, which are presented in the following section.

- Prepare your environment

```
python3 -m venv venv
source venv/bin/activate

pip install --upgrade pip
```

- Install the required and necessary packages 📦

```
# requirements.txt
HuggingFace
torch
transformers
accelerate
```

```
torchvision
torchaudio
```

```
pip install -r requirements.txt
```

- Once the environment is ready, just copy and run these two simple apps!

> *The 1st code constructs simply a chat prompt using the tokenizer's template to enable tool-calling behavior in response to the user's query about Boston weather. After tokenizing the prepared input and moving it to the selected device, the model generates an output sequence.*
>
> *The 2nd script performs a complete machine learning inference pipeline using a pre-trained LLM. It then loads the IBM Granite 4.0 (350M parameter) model and tokenizer, prepares a user prompt for a research lab location query, and generates a response from the model.*

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import json
import os # Import os for file system operations

# --- Device Detection and Selection ---
# Automatically determine the best device available (CUDA > MPS > CPU)
if torch.cuda.is_available():
    device = "cuda"
elif hasattr(torch.backends, 'mps') and torch.backends.mps.is_available():
    # MPS (Metal Performance Shaders) is the accelerator for Apple Silicon (M1/
    device = "mps"
else:
    device = "cpu"

print(f"Selected device: {device}")
# --- End Device Detection ---


model_path = "ibm-granite/granite-4.0-350M"
tokenizer = AutoTokenizer.from_pretrained(model_path)

# Pass the automatically determined device to device_map
# The model will now load onto the CPU (or MPS/CUDA if available)
model = AutoModelForCausalLM.from_pretrained(model_path, device_map=device)
```

```
model.eval()

tools = [
    {
        "type": "function",
        "function": {
            "name": "get_current_weather",
            "description": "Get the current weather for a specified city.",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {
                        "type": "string",
                        "description": "Name of the city"
                    }
                },
                "required": ["city"]
            }
        }
    }
]

# change input text as desired
chat = [
    { "role": "user", "content": "What's the weather like in Boston right now?"
]

chat = tokenizer.apply_chat_template(chat, \
                                     tokenize=False, \
                                     tools=tools, \
                                     add_generation_prompt=True)

# tokenize the text and move to the selected device
input_tokens = tokenizer(chat, return_tensors="pt").to(device)

# generate output tokens
output = model.generate(**input_tokens,
                        max_new_tokens=100)

# decode output tokens into text
output = tokenizer.batch_decode(output)

# --- Save output to file in Markdown format ---
output_dir = "./output"
output_file = os.path.join(output_dir, "output.md")

# Create the output directory if it doesn't exist.
# The exist_ok=True argument prevents an error if the directory already exists.
try:
    os.makedirs(output_dir, exist_ok=True)

    # Write the output to the Markdown file
    with open(output_file, "w", encoding="utf-8") as f:
```

```
            # The output from batch_decode is a list, we take the first item (the g
            f.write(output[0])

        # Confirmation message
        print(f"\nModel output saved successfully to {output_file}")

        # Optionally print the content to the console for immediate review
        print("\n--- Generated Content ---\n")
        print(output[0])
        print("\n-------------------------")

    except Exception as e:
        print(f"\nAn error occurred while trying to save the output file: {e}")
    # --- End File Saving Logic ---
```

. . .

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import os # Import os for file system operations

# --- Device Detection and Selection ---
# Automatically determine the best device available (CUDA > MPS > CPU)
if torch.cuda.is_available():
    device = "cuda"
elif hasattr(torch.backends, 'mps') and torch.backends.mps.is_available():
    # MPS (Metal Performance Shaders) is the accelerator for Apple Silicon (M1/
    device = "mps"
else:
    device = "cpu"

print(f"Selected device: {device}")
# --- End Device Detection ---


model_path = "ibm-granite/granite-4.0-350M"
tokenizer = AutoTokenizer.from_pretrained(model_path)

# Pass the automatically determined device to device_map
model = AutoModelForCausalLM.from_pretrained(model_path, device_map=device)
model.eval()

# change input text as desired
chat = [
    { "role": "user", "content": "Please list one IBM Research laboratory locat
```

```
]

chat = tokenizer.apply_chat_template(chat, tokenize=False, add_generation_promp

# tokenize the text and move to the selected device
input_tokens = tokenizer(chat, return_tensors="pt").to(device)

# generate output tokens
output = model.generate(**input_tokens,
                        max_new_tokens=100)

# decode output tokens into text
output = tokenizer.batch_decode(output)

# --- Save output to file in Markdown format ---
output_dir = "./output"
output_file = os.path.join(output_dir, "output.md")

# Create the output directory if it doesn't exist.
try:
    os.makedirs(output_dir, exist_ok=True)

    # Write the output to the Markdown file
    with open(output_file, "w", encoding="utf-8") as f:
        # The output from batch_decode is a list, we take the first item (the g
        f.write(output[0])

    # Confirmation message
    print(f"\nModel output saved successfully to {output_file}")

    # Optionally print the content to the console for immediate review
    print("\n--- Generated Content ---\n")
    print(output[0])
    print("\n-------------------------")

except Exception as e:
    print(f"\nAn error occurred while trying to save the output file: {e}")
# --- End File Saving Logic ---
```

. . .

You'll get these outputs 📄

```
<|start_of_role|>system<|end_of_role|>You are a helpful assistant with access t
```

```
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "get_current_weather", "description":
</tools>

For each tool call, return a json object with function name and arguments withi
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>. If a tool does not exist in the provided list of tools, notify th
<|start_of_role|>user<|end_of_role|>What's the weather like in Boston right now
<|start_of_role|>assistant<|end_of_role|><tool_call>
{"name": "get_current_weather", "arguments": {"city": "Boston"}}
</tool_call><|end_of_text|>


======


<|start_of_role|>system<|end_of_role|>You are a helpful assistant. Please ensur
<|start_of_role|>user<|end_of_role|>Please list one IBM Research laboratory loc
<|start_of_role|>assistant<|end_of_role|>IBM Research Laboratory: Cambridge Res
```

Et voilà 🥇

Conclusion

The Granite 4.0 model family represents a pivotal shift toward highly efficient, enterprise-grade AI, redefining performance by focusing on accessibility rather than sheer scale. A key strength lies in its innovative hybrid Mamba/Transformer architecture, which significantly reduces memory requirements — often by over 70% — enabling robust inference on **rudimentary and affordable hardware**, including consumer-grade GPUs and edge devices. Crucially, as an **open-source** offering under the Apache 2.0 license, Granite 4.0 models empower developers with complete operational sovereignty, allowing for deep customization, on-premise deployment for enhanced data privacy, and full transparency. This combination of superior efficiency and open governance lowers the barrier to entry, democratizing advanced AI for complex workflows like RAG and function calling, while ensuring the control and trust necessary for real-world business adoption.

Thank's for reading 🧠

Links

- Granite 4 Models on Hugging Face: https://huggingface.co/collections/ibm-granite/granite-40-nano-language-models
- Granite 4 Models on Ollama: https://ollama.com/library/granite4
- Granite Documentation: https://www.ibm.com/granite/docs/models/granite
- Granite Playground: https://www.ibm.com/granite/playground
- Granite Cookbook: https://www.ibm.com/granite/docs/use-cases/all-cookbooks
- Granite on Docker Hub: https://hub.docker.com/r/ai/granite-4.0-h-nano
- Granite 4 web gpu: https://hub.docker.com/r/ai/granite-4.0-h-nano
- IBM Granite Communities: https://github.com/ibm-granite-community

Llm | Llm Applications | Granite | Hugging Face | Ollama

Follow

Written by Alain Airom (Ayrom)

708 followers · 969 following

IT guy, IBMer... sharing my hands-on experiences and technical subjects of my interest (IBM or not). A bit "touche à tout"!

Responses (1)

Bgerby

What are your thoughts?

Romualdo Gobbo
2 days ago

Thank a lot for this quick guide!

3 1 reply Reply

More from Alain Airom (Ayrom)

Alain Airom (Ayrom)

First Hands-On Experience with Apple Containers!

A very first test with Apple containers!

Sep 8 100 4

In Artificial Intelligence in Plain English by Alain Airom (Ayrom)

The Secret to Efficient RAG: A Step-by-Step Guide to Chunking and Counting Your Vectors

Debunking the vetor count estimation for documents

Oct 2 22

In Artificial Intelligence in Plain English by Alain Airom (Ayrom)

📣 Just announced: IBM Granite-Docling: End-to-end document understanding with one tiny model

Another exciting feature with Docling!

Sep 17 277 3

In Artificial Intelligence in Plain English by Alain Airom (Ayrom)

Testing Ollama Web Search 🔍 and a Thinking Model 💭

Testing Ollama Web Search with gpt-oss

Sep 27 62 1

See all from Alain Airom (Ayrom)

Recommended from Medium

Alain Airom (Ayrom)

Using Chonkie

Testing Chonkie 🧪

Oct 24 56

In Towards AI by Teja Kusireddy

We Spent $47,000 Running AI Agents in Production. Here's What Nobody Tells You About A2A and MCP.

Multi-agent systems are the future. Agent-to-Agent (A2A) communication and Anthropic's Model Context Protocol (MCP) are revolutionary. But...

Oct 16 1.6K 37

In Level Up Coding by Hassan Nauman

8 Python Libraries That Replace Entire Paid Tools

Free, open, and shockingly powerful.

Oct 24 247 1

In Coding Nexus by Code Coup

Claude Desktop Might Be the Most Useful Free Tool You'll Install This Year

I didn't expect much when I first saw the announcement for Claude Desktop. Another AI wrapper, I thought. Maybe with a shiny UI.

Oct 23 297 12

In Artificial Intelligence in Plain English by DhanushKumar

Nanonets-OCR2

Nanonets-OCR2 is a family of image → markdown / image → text models (a visual-language model, VLM) built to convert complex documents into...

Oct 21 11

In Data Science Collective by Ida Silfverskiöld

Agentic AI: Single vs Multi-Agent Systems

Building with a structured data source in LangGraph

3d ago 378 9