AI4Bharat/Airavata Quantization + FastAPI Backend

AIM:

The aim of this project is to efficiently quantize the AI4Bharat/Airavata large language model for optimized performance on both CPU and GPU**.** The project also involves developing a FastAPI backend service to serve the quantized model for inference, with a focus on reducing latency and increasing throughput without compromising generation quality. All inference metrics such as latency and throughput are to be captured and evaluated, using a single-machine setup.

Step-by-Step Procedure (GPU Quantization +FastAPI Backend):

1. Install required libraries

Installed essential libraries for quantization, serving and inference:

- fastapi, uvicorn for the backend
- nest-asyncio and pyngrok for running FastAPI in Google Colab
- transformers, accelerate, and bitsandbytes for model loading and 4-bit quantization

2. BitsAndBytes for 4-Bit Quantization

BitsAndBytesConfig setup with:

- load_in_4bit=True to enable 4-bit model loading
- bnb_4bit_quant_type="nf4" to use normal float-4 precision
- bnb_4bit_compute_dtype=torch.float16 to use FP16 for computation
- bnb_4bit_use_double_quant=True to further compress weights using double quantization

This configuration allows reduced memory usage and better inference speed on GPU.

3. Load the Model in 4-Bit on GPU

Loaded the ai4bharat/airavata model using transformers.AutoModelForCausalLM with:

- The 4-bit quantization configuration from above
- device_map="auto" to automatically place model layers on the available CUDA device (GPU)

The tokenizer was loaded normally.

4. Define a Prompt Formatting Function

Since the Airavata model uses a chat format, created a helper function directly based on the official prompt formatting strategy recommended by Ai4Bharat on Hugging Face**.** The helper function is used to wrap user prompts inside special tokens like <|user|> and <|assistant|>, simulating a conversational setup.

5. Create FastAPI Backend

Created a FastAPI app with a /generate POST endpoint. The steps inside this endpoint:

1. Start timing for latency measurement.
2. Tokenize the formatted user prompt and send it to GPU (.to("cuda")).
3. Generate the response using .generate() with temperature sampling.
4. Decode the output text.
5. Measure:
    - **Latency** (in seconds)
    - **Tokens generated**
    - **Throughput** = tokens / latency
6. Return these metrics and the generated text as JSON

6. FastAPI on Colab using Ngrok

Since Colab can't expose ports directly, hence:

- Applied nest_asyncio to allow event loops in notebooks.
- Used pyngrok.connect() to expose the FastAPI server on an HTTPS public URL.
- Launched the app using uvicorn.run().

7. Inference via Swagger UI

Accessed the public URL (e.g., https://xxxx.ngrok-free.app/docs) to interact with the /generate endpoint using Swagger UI and made requests directly with Hindi prompts.

8. Performance Metrics

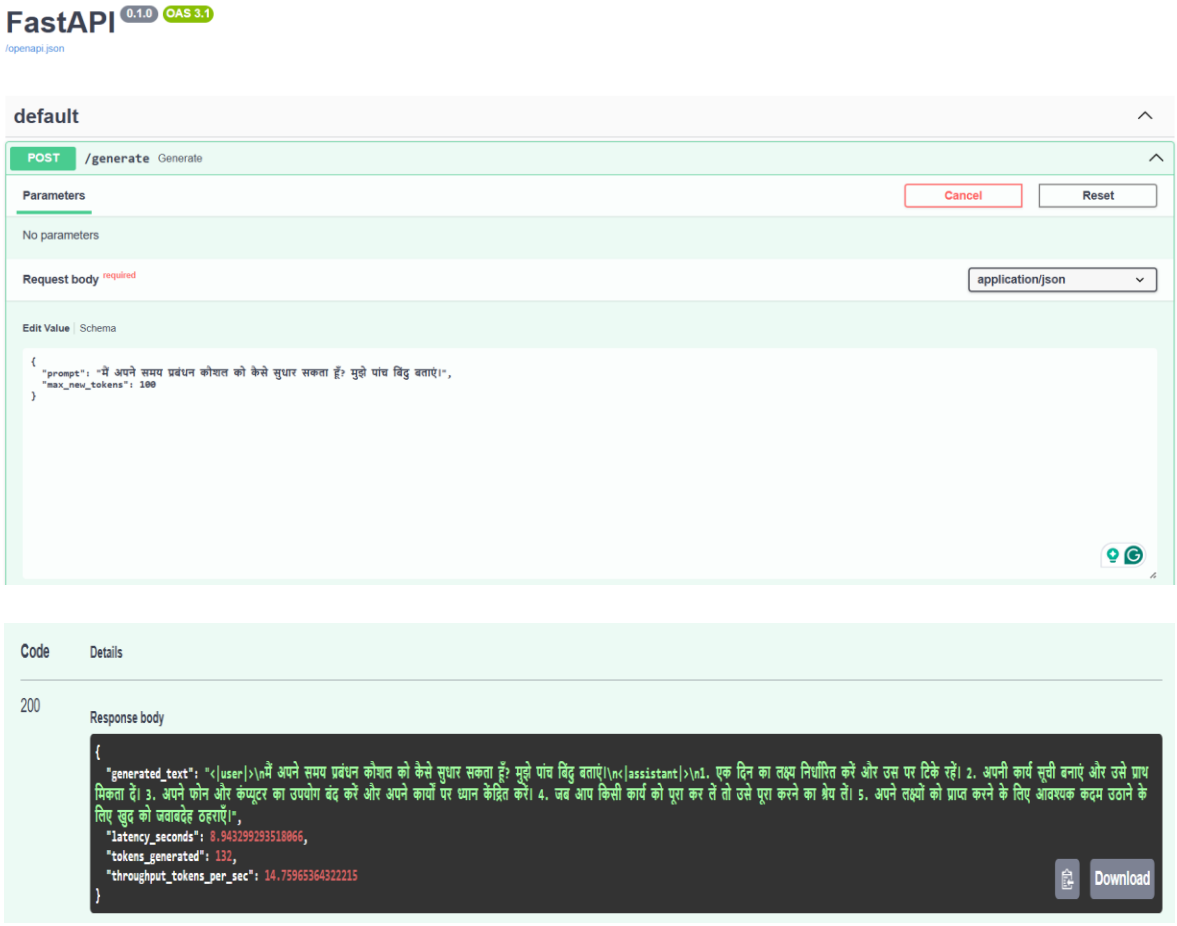
From the request, obtained:

- The actual generated text
- The latency (how long it took to generate)
- The number of tokens generated
- The throughput (speed in tokens per second)

These metrics help evaluate how well the quantized model performs on GPU.

Output:

- Inference was successfully performed on the quantized GPU model using the FastAPI backend.
- The output was obtained via the Swagger UI interface (/generate endpoint).
- The response was downloaded as a JSON file directly from the Swagger UI.



Step-by-Step Procedure (CPU Quantization):

1. **Aim:** Quantize and run inference on **CPU**.
2. **Steps:**
    - Switch device to CPU.
    - Load the full model on CPU.
    - Apply quantization

Issues Faced:

1. **Hardware Limitation:**
    - Airavata is a very large model.
    - Even on skipping quantization and load it directly on CPU, the local machine or Google Colab RAM isn't enough**.** This led to out-of-memory errors or crash during model load.