**Project Title:**

Audio Signal Processing Assistant Agent using IBM Granite AI and IBM Cloud

**Project Description:**

This project presents an AI-powered assistant agent designed to support engineers

and students in the field of audio signal processing. The agent utilizes

IBM Granite, a large language model (LLM), via IBM Watsonx.ai and is deployed on

the IBM Cloud Lite platform. It interprets natural language queries related to

audio electronics, such as amplifiers, filters, and preamplifier circuits,

and provides intelligent suggestions for design improvements, noise reduction,

and debugging.

By acting as an intelligent assistant, it can answer complex questions like:

"Why is my op-amp circuit producing distortion?"

"How can I design a low-pass filter for 10kHz cut-off?"

"How do I reduce 50Hz hum in my audio amplifier?"

The assistant provides guidance on:

Filter design and component selection

Circuit stability and gain issues

Grounding and shielding techniques

Common fault diagnosis in analog audio systems

Built without manual model training, the agent leverages agentic AI principles

and prompt engineering to offer real-time expert suggestions, making it ideal

for educational use and rapid prototyping in audio electronics.

**STEPS:**

1.creating the account in the ibm cloud and enabling student version lite version

2.using watsonx.ai studio

3.create the storage with free type in london location

4.next crate the project

5.add cloud storage object for the project storage in the ibm cloud

**Next Steps: Add and Use IBM Granite AI Model**

**🔹 Step 1: Add the Foundation Model to Your Project**

1. Click on the **“Assets”** tab (next to “Overview”).
2. Click **“Add to project”** → Select **“Foundation model”**
   * If you don't see it, scroll down or click “View all assets” first.
3. In the search bar, type: Granite
4. Choose one of these models:
   * Granite-13b-chat-v2 *(best for question answering and conversations)*
   * Or any version like granite-13b-instruct-v1 if the above isn’t listed
5. Click **“Add to project”**

**🔹 Step 2: Open the Prompt Lab**

1. Once the Granite model is added, it will appear under **Assets**.
2. Click the model name (e.g., Granite-13b-chat).
3. Select **“Open in Prompt Lab”** from the top-right.

**🔹 Step 3: Start Testing Prompts**

Paste one of the following sample queries into the prompt box:

| **Sample Prompt** | **Purpose** |
| --- | --- |
| "Why is my audio amplifier producing noise?" | Debugging tip |
| "Design a low-pass filter with 10kHz cutoff." | Filter design |
| "Suggest how to reduce distortion in an op-amp audio circuit." | Circuit optimization |
| "What causes 50Hz hum in preamp circuits?" | Grounding/noise analysis |

Click **Submit** or **Run** and wait for the Granite model to reply with its intelligent output.

**Optional: Save Good Prompts**

You can click **“Save”** on useful prompt sessions so you can reuse them later.

**After Testing: Next Phase**

Once you confirm the model is responding well, you can proceed to:

* Create a backend (IBM Cloud Function) that sends user queries to Granite
* Connect it to a CLI or simple frontend

Great! You're at the right stage inside your IBM Watsonx.ai project for **“Audio Signal Processing Assistant Agent”**. Here's your **next step-by-step guide** from here:

**✅ STEP 1: Click on New asset**

In the top-right corner of the screen (as shown in your image), click the blue **“New asset”** button.

**✅ STEP 2: Select Notebook**

In the dropdown menu that appears:

* Choose **Notebook**  
  (This will allow you to write Python code using IBM’s AI models like Granite and integrate audio signal queries.)

**✅ STEP 3: Configure the Notebook**

A setup page will appear. Configure as follows:

**1. Name:**

AudioSignalAssistantNotebook

**2. Runtime environment:**

Choose:

* **Default Python 3.x runtime** (if available)
* Or choose: Python 3.x + Spark 3.4 with 1 vCPU, 4GB RAM (Lite plan limits)

Then click **Create**.

**✅ STEP 4: Inside the Notebook**

Once the notebook opens:  
You will write Python code to:

1. Connect to **IBM Granite AI model**
2. Input audio circuit issues as prompts
3. Return suggestions using generative AI

I’ll give you the sample code as soon as your notebook is ready.

Excellent! You're now inside your IBM Watson Studio notebook for the **Audio Signal Processing Assistant** project. Let’s start building it step by step.

**🔧 Step 1: Install & Import Required Libraries**

Paste the following code in the **first cell** and run it (Shift + Enter):

python

CopyEdit

# Install IBM Watson SDK

!pip install ibm-watson -q

!pip install ibm-cloud-sdk-core -q

Then, in the **next cell**, paste this and run:

python

CopyEdit

from ibm\_watson import AssistantV2

from ibm\_cloud\_sdk\_core.authenticators import IAMAuthenticator

import json

**🔑 Step 2: Set Up API Credentials**

Now, paste the following **into the next cell**, and **fill in your actual credentials** from IBM Cloud:

python

CopyEdit

# Replace with your actual API Key and URL from IBM Watson Assistant service

API\_KEY = 'your-ibm-api-key-here'

ASSISTANT\_ID = 'your-assistant-id-here'

URL = 'your-service-url-here'

# Authenticate

authenticator = IAMAuthenticator(API\_KEY)

assistant = AssistantV2(

version='2021-06-14',

authenticator=authenticator

)

assistant.set\_service\_url(URL)

⚠️ **Where do I find this info?**  
Go to your IBM Cloud dashboard → Watson Assistant → click on your Assistant instance → copy API Key, Assistant ID, and URL.

**🔄 Step 3: Create a Session**

Paste and run this:

python

CopyEdit

# Create a session with the assistant

session\_response = assistant.create\_session(

assistant\_id=ASSISTANT\_ID

).get\_result()

session\_id = session\_response['session\_id']

print("Session created:", session\_id)

**💬 Step 4: Ask Questions to IBM Granite (Watson Assistant)**

Now you can test by sending a prompt:

python

CopyEdit

# Ask your AI Assistant a question

response = assistant.message(

assistant\_id=ASSISTANT\_ID,

session\_id=session\_id,

input={

'message\_type': 'text',

'text': 'Why is my audio circuit output distorted at high frequencies?'

}

).get\_result()

print(json.dumps(response, indent=2))

Sure, here's how you can get your **API Key**, **URL**, and **Assistant ID** from **IBM Watson Assistant on IBM Cloud** step-by-step:

**🔧 Step-by-Step to Get IBM Watson Assistant Credentials**

**🪄 Step 1: Go to IBM Cloud Console**

1. Open: https://cloud.ibm.com/
2. Log in to your IBM Cloud account.

**🤖 Step 2: Open Watson Assistant Service**

1. On the **dashboard**, look under **“Resource List”** or **“Services”**.
2. Click on the service name (e.g., Watson Assistant).
   * If you haven’t created one yet, click “**Create Resource**” > Search “Watson Assistant” > click it > click **Create**.

**🔐 Step 3: Get the API Key and URL**

1. Inside your Watson Assistant service page, click on **“Service credentials”** from the left sidebar.
2. If there’s no credential, click **“New Credential”**.
3. Click the **“View credentials” (eye icon)**.

You'll see a JSON block like:

json

CopyEdit

{

"apikey": "YOUR\_API\_KEY\_HERE",

"url": "https://api.eu-gb.assistant.watson.cloud.ibm.com"

}

✅ **Copy**:

* apikey → Use it in your code as apikey
* url → Use it in your code as url

**🆔 Step 4: Get the Assistant ID**

1. Go to the **“Assistants”** tab in the left menu.
2. Click on your Assistant (or create a new one).
3. Go to the **“Settings”** tab (⚙️).
4. You’ll find the **Assistant ID** there — copy it.

✅ Use this as assistant\_id in your code.

**✅ Recap of What You Need to Copy:**

| **Key** | **Where to Find** |
| --- | --- |
| apikey | Service credentials tab |
| url | Service credentials tab |
| assistant\_id | Inside your Assistant > Settings |

Once you copy these, paste them into your notebook code like this (don't share these keys publicly):

python

CopyEdit

apikey = 'your-real-api-key-here'

url = 'your-real-url-here'

assistant\_id = 'your-real-assistant-id-here'

**✅ Your IBM Watson Assistant credentials:**

* **API Key:** Visible ✔️
* **URL:** Visible ✔️

**✅ Next Step: Get your Assistant ID**

Here’s how:

**🧭 Steps to Get Assistant ID:**

1. Click on **“Launch watsonx Assistant”** (the blue button you see).
2. In the opened Watson Assistant interface:
   * Go to the **“Assistants”** tab on the left panel.
   * Click **“Create Assistant”** if you haven’t already (name it something like AudioSignalBot).
   * After the assistant is created, click on it.
   * Now go to the **“Settings”** ⚙️ tab inside the Assistant.
   * You will see an **Assistant ID** — copy it.

Once you have all three:

| **Parameter** | **Source** |
| --- | --- |
| API Key | From **Service credentials** tab |
| URL | From **Service credentials** tab |
| Assistant ID | From **Watson Assistant settings** |

Would you like me to help you:

* ✅ Test a basic conversation in a Python Notebook?
* ✅ Connect this assistant to your **Audio Signal Processing Assistant** project?

Let me know once you have the **Assistant ID** and I’ll give you the full code to test it!

Top of Form

Tools

**🔧 Step-by-Step: Creating Your Assistant**

1. **Assistant name**:  
   Enter:

arduino

CopyEdit

Audio Signal Bot

1. **Description (optional)**:  
   You can leave this blank or enter:

css

CopyEdit

A virtual assistant for audio signal processing queries and tasks.

1. **Assistant language**:  
   Keep as: English (US)
2. ✅ Then click **Next** (top-right of the page) — it will become clickable after filling the required fields.

**🎯 What happens next:**

* You’ll go through “Personalize” and “Customize” steps.
* In the final stage, you’ll **see your Assistant ID**.

Once created, **go to the Settings tab of your assistant** and copy the Assistant ID.

🛠️ **After that**, I’ll give you:

* ✅ Python code to test the assistant using API Key, URL, and Assistant ID.
* ✅ Help you integrate it with your Audio Signal Processing project.
* **✅ How to Get API Key, URL, and Assistant ID**
* **🔹 Step 1: Go to Assistant Settings**
* On the left sidebar, **click your Assistant name** (e.g., *AudioSignal AI*).
* Then click **“Settings”** (usually a gear icon).
* **🔹 Step 2: Open “API Details” or “Deployment”**
* Look for a section like **"API Details"**, **“Deploy”**, or **“Assistant Details”**.
* Copy these 3 things:

| * **Key** | * **Where to Find** | * **Notes** |
| --- | --- | --- |
| * **Assistant ID** | * Under "Assistant details" or "API reference" | * Used to identify your bot |
| * **API Key** | * Under "Service credentials" or "Manage" | * Used for authentication |
| * **Service URL** | * Usually starts with https://api.us-south.assistant... | * Used to connect to service |

* **🔐 If you don’t see the API key**
* Go to IBM Cloud Resource List.
* Find your **watsonx Assistant service**.
* Click it → Go to **"Service Credentials"** tab.
* Create new credential → Copy **apikey**, **url**, and other values.
* **Training Plan for Your Project: *Audio Signal Processing Assistant Agent***
* **🔰 Phase 1: Basics of IBM Watsonx Assistant**

| * **Topics** | * **Learning Outcome** |
| --- | --- |
| * What is IBM Watsonx Assistant? | * Understand what a chatbot assistant is and how it's used. |
| * Types of Assistants: Dialog-based vs Action-based | * Choose correct design model for your project. |
| * Overview of Watsonx UI: Actions, Intents, Variables | * Familiarize yourself with the platform. |

* **✅ Tasks:**
* ✅ Create a Watsonx Assistant on IBM Cloud Lite
* ✅ Customize the persona and deployment
* ✅ Create your first Action (e.g., “Greet”, “Help”, “Ask audio type”)
* **🔰 Phase 2: Dialog Flow Design (Core of the Project)**

| * **Topics** | * **Learning Outcome** |
| --- | --- |
| * Rule-based and intent-based actions | * Learn to handle user queries effectively |
| * Conditional logic in responses | * Process different types of audio requests |
| * Variables & Slots | * Collect and store audio parameters like file type, duration, filter type |

* **✅ Tasks:**
* ✅ Build a flow like:
* Greet the user
* Ask for audio format (WAV, MP3)
* Ask what processing is needed (noise removal, compression, enhancement)
* Output message or pass control to backend
* **🔰 Phase 3: Backend Integration & API Handling**

| * **Topics** | * **Learning Outcome** |
| --- | --- |
| * How Watsonx sends messages to backend via Webhooks/API | * Let the agent process real audio signals |
| * Connect to Flask / Node.js / Python app for signal processing | * Process audio using scipy, librosa, etc. |
| * Return response to Watson Assistant | * Display processed info or result |

* **✅ Tasks:**
* ✅ Create backend using Flask (or Node.js)
* ✅ Build audio processing module:
* Noise removal using scipy / librosa
* Plot waveform (optional)
* Save result
* ✅ Connect backend API to Watsonx using webhook
* **🔰 Phase 4: Deployment & Testing**

| * **Topics** | * **Learning Outcome** |
| --- | --- |
| * Deploying on IBM Cloud Functions or IBM App Runtime | * Free deployment using IBM Lite tier |
| * Using API key, Assistant ID for secure access | * Securely invoke Watson assistant from frontend |
| * Test with Web Chat / Mobile App / Web Embed | * Deliver final assistant to end users |

* **✅ Tasks:**
* ✅ Deploy API to IBM Cloud or Render.com
* ✅ Embed chatbot in a simple webpage
* ✅ Test from mobile/laptop
* **🔬 Bonus: AI-Powered Signal Suggestions**

| * **Ideas** | * **Tools** |
| --- | --- |
| * Use IBM Granite (AI) to detect user mood or recommend best signal filters | * Watsonx.ai, LangChain |
| * Upload audio and auto-detect issues | * AI Audio Quality Analysis using ML |
| * Store user interactions for feedback learning | * IBM Cloudant or MongoDB |

* **🧑‍🎓 Skills You Will Gain**
* Chatbot Design (Watsonx)
* Audio Signal Processing (Python)
* API/Webhook Integration
* IBM Cloud Lite Deployment
* Real-time Conversation Handling

Bottom of Form

**Project Name:**

**Audio Signal Processing Assistant Agent**

**🗣️ Conversation Steps (Dialog Flow)**

**✅ Step 0: Greeting**

* **Trigger**: User says “hi”, “hello”, or opens chat
* **Assistant says**:

Hello! I’m your Audio Signal Processing Assistant Agent.  
I can help you with noise, distortion, filter design, or amplifier issues.  
How can I assist you today?

**✅ Step 1: Identify User Problem**

* **Assistant asks**:

What type of audio problem are you facing?  
(You can choose or type your issue)

* **User Choices** (Multiple choice or text):
  + "My amplifier is producing noise"
  + "I need help with op-amp distortion"
  + "How to design a filter"
  + "Other"
* **Set Variable**: problem\_type

**✅ Step 2: Ask for Audio Type**

* **Assistant asks**:

What is the type of your input audio signal?  
(e.g., voice, music, sine wave, etc.)

* **User Input**: Free Text
* **Set Variable**: audio\_type

**✅ Step 3: Ask for Signal Format**

* **Assistant asks**:

What is your audio file format? (MP3 / WAV / FLAC / other)

* **User Input**: Multiple choice or free text
* **Set Variable**: file\_format

**✅ Step 4: Provide Diagnosis or Suggestions**

* **Use conditional logic** on problem\_type, such as:

**If problem\_type = “My amplifier is producing noise”**, reply:

This may be due to poor grounding, power supply ripple, or input impedance mismatch.  
🛠 Try:

* Use a bypass capacitor (e.g., 0.1µF)
* Shield sensitive parts
* Use differential inputs

**If problem\_type = “op-amp distortion”**, reply:

Distortion may be caused by:

* Exceeding op-amp slew rate
* Wrong feedback network
* Operating near supply rails  
  🔧 Use low-distortion op-amps like TL072, and ensure headroom in design.

**If problem\_type = “How to design a filter”**, reply:

What type of filter do you want to design?

* Low-pass
* High-pass
* Band-pass
* Notch

→ Then proceed with formula suggestion based on filter type.

**✅ Step 5: Offer External Help or Processing (Optional)**

* **Assistant says**:

Would you like me to process an audio sample or suggest a circuit diagram?

→ If user says "yes", redirect to a backend API.

Example:

* Send JSON:

json

CopyEdit

{

"file\_format": "WAV",

"problem": "noise",

"audio\_type": "voice"

}

→ The backend could return a waveform analysis or text summary.

**✅ Step 6: Ending / Handoff**

* **Assistant says**:

Hope that helped! If you have another issue or want to process another file, just let me know.  
I’m always here to assist you with your circuits and audio systems 🎧🔧

**🔁 Optional Loop:**

You can loop back to Step 1 to allow multiple queries in the same session.

**💡 Tips to Implement in Watsonx Assistant:**

* Use **"If/Else conditions"** for problem\_type
* Use **variables**: audio\_type, file\_format, filter\_type, etc.
* You can **link this to a backend API** later when needed

**PROJECT COMPLETION CHECKLIST**

**✅ 1. Test Your Agent Interactions**

Ensure the agent responds correctly to questions like:

* "Why is my op-amp circuit producing noise?"
* "How can I reduce distortion?"
* "What is the best filter for a 60 Hz hum?"

Make sure the flow:

* Collects file\_format, problem, and audio\_type
* Gives suggestions (using If/Else conditions or dialog branching)

**✅ 2. Enhance Responses with Audio Design Suggestions**

Here’s what your assistant can say based on user input:

| **Problem** | **Suggestion** |
| --- | --- |
| noise | “Try adding bypass capacitors and ensure proper grounding. Also, isolate analog and digital grounds.” |
| distortion | “Check the op-amp's slew rate, avoid overdriving inputs, and keep signals within linear operating range.” |
| low gain | “Ensure feedback resistor values are optimized and input impedance is suitable for the source.” |

✅ You can create a **lookup table** inside the action flow or even a switch case or If/Else logic.

**✅ 3. Add UI Personalization**

If you’re in the Watsonx Assistant UI designer section:

* 🧑‍💻 **Add your logo or project name**: “Audio Signal Assistant”
* 🎨 Change theme, fonts, and bubble colors to match your domain (telecom, circuits, etc.)
* 💬 Set a welcome message:

“Hi, I’m your Audio Signal Processing Assistant. I can help with amplifier noise, filter design, grounding, and more. Ask me anything!”

**✅ 4. Enable Web Chat Preview & Share Link**

Do this to test or submit:

* Click **Preview**
* Interact with the agent
* Click **Deploy** → Share the **public URL** for demo or report submission

**✅ 5. (Optional but Strongly Recommended): Add Watsonx AI/Granite**

To showcase **AI integration**, do either:

**Option A – Watsonx AI Prompt**

1. Go to Watsonx.ai → Create a new **Prompt Lab**
2. Choose Granite-13b or Granite-coder
3. Use a prompt like:

I'm debugging an audio amplifier with high noise. What could be the cause and how can I fix it?

1. Copy the response
2. Mention this in your documentation:

“We used IBM’s Granite AI to generate expert suggestions dynamically for amplifier noise problems.”

**Option B – Integrate via API (advanced)**

Expose a Flask or Cloud Function that calls Granite, then connect via Watsonx Assistant webhook (if time permits).

**✅ 6. Final Deliverables (for submission)**

Here’s what to prepare:

| **Item** | **Description** |
| --- | --- |
| Project Title | **Audio Signal Processing Assistant Agent** |
| Project Description | An AI agent that assists in designing/debugging audio circuits by identifying common issues such as noise, distortion, and gain problems, and offering suggestions. |
| IBM Services Used | Watsonx Assistant (for chat), Watsonx.ai (Granite AI, optional), IBM Cloud Lite |
| Input Format | file\_format, problem, audio\_type |
| Example Interaction | “I have a WAV file with a noise issue in voice recording.” |
| Output Example | “Check for improper grounding and power supply ripple. Use bypass capacitors near the op-amp.” |
| Chatbot Link | (Share your deployed chatbot URL) |
| Screenshots | UI designer, test conversation, assistant flow editor |

**Final Goal**

👉 **Host your chatbot (HTML + Watson Assistant)** as a public webpage using **IBM Cloud**.

**💡 You have 2 recommended options to host it:**

**### ✅ Option 1: IBM Cloud Object Storage + Static Website Hosting (Best for HTML + JS)**

This is **easy, free**, and perfect for hosting simple HTML files like yours.

**📦 Steps:**

**1. Zip your HTML file**

Include only:

* Your main .html file (with chatbot script)
* Any CSS/JS files if needed

Name it like: chatbot-site.zip

**2. Create an Object Storage Bucket**

1. Go to [IBM Cloud Console](https://cloud.ibm.com/)
2. Click **Create resource**
3. Search for **Object Storage**
4. Choose **Cloud Object Storage** and create an instance (Lite plan is free)
5. After creating, go to **Buckets > Create bucket**
6. Name: watson-chatbot-bucket
7. Type: **Public** or **Public with CORS** (if needed for cross-origin)

**3. Enable Static Website Hosting**

1. Inside your bucket, go to **"Configuration"** tab
2. Turn ON **Static Website Hosting**
3. Set:
   * **Index Document**: index.html
   * **Error Document**: 404.html (or leave blank)
4. Save it

**4. Upload your HTML zip**

1. Go to **Objects > Upload**
2. Upload and extract your chatbot-site.zip
3. Make sure index.html is in root

**5. Get your public URL**

* Go to **Bucket > Configuration > Static Website**
* You’ll see a URL like:  
  https://<bucket-name>.<region>.objectstorage.appdomain.cloud/index.html

Open it — your Watson chatbot will be live!

**✅ Option 2: IBM Cloud Code Engine or App Hosting (Advanced)**

If you want dynamic backend + frontend, use IBM Cloud App Hosting (not needed for guide-only bots). Let me know if you want this path.

**📌 Example: Minimal index.html for Watson Assistant Hosting**

<!DOCTYPE html>

<html>

<head>

<title>Watson Audio Assistant</title>

</head>

<body>

<h1>Welcome to the Audio Signal Processing Assistant</h1>

<div id="watson-assistant"></div>

<script>

window.watsonAssistantChatOptions = {

integrationID: "0a9eb86b-d9a5-48ac-9947-e31c5dccc546",

region: "eu-gb",

serviceInstanceID: "1e92c7d5-b8dd-4dba-8460-592e57ed1127",

onLoad: async (instance) => { await instance.render(); }

};

setTimeout(function(){

const t=document.createElement('script');

t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +

(window.watsonAssistantChatOptions.clientVersion || 'latest') +

"/WatsonAssistantChatEntry.js";

document.head.appendChild(t);

});

</script>

</body>

</html>

The proposed system aims to assist engineers and hobbyists in diagnosing and designing audio circuits effectively by leveraging IBM Cloud services and advanced AI models. This will be achieved through the following components:

**1. Knowledge Base Integration:**

* Use IBM Granite AI to incorporate domain knowledge of audio signal processing.
* Train the assistant with queries related to preamps, op-amps, filters, and grounding techniques.

**2. Chatbot Design (Watson Assistant):**

* Build an interactive agent using IBM Watson Assistant.
* Implement custom intents like “amplifier noise,” “reduce distortion,” “op-amp gain issue,” etc.
* Enable context switching for multi-turn conversations.

**3. Web UI & Interface:**

* Design a responsive static web interface using HTML, CSS, and JS.
* Include a typing animation intro, chatbot docked at bottom-right, and floating arrow for guidance.
* Background themed with audio circuits.

**4. Deployment Strategy:**

* Host the solution on IBM Cloud Object Storage as a static website.
* Use IBM Cloud Functions if dynamic updates or logging is required.
* Ensure mobile-friendly layout and smooth UX across devices.

**5. Evaluation & Feedback:**

* Collect feedback from users on accuracy and helpfulness.
* Refine chatbot responses using Watson Assistant logs and user interaction data.
* Expand scope to cover additional audio circuit types and real-time debugging tools.

**System Requirements:**

* **Cloud Platform**: IBM Cloud Lite
* **AI Model**: IBM Granite AI
* **Frontend**: HTML, CSS, JavaScript (responsive, mobile-friendly)
* **Backend (optional)**: IBM Cloud Functions (for dynamic interaction if needed)
* **Chatbot Framework**: IBM Watson Assistant
* **Deployment**: IBM Cloud Object Storage (Static Hosting)

**Libraries & Tools Used:**

* IBM Watson Assistant – for building conversational agent
* Granite AI Model – for deep learning-based query responses
* jQuery – for DOM manipulation and animations
* Typed.js – for typing animation effect
* Bootstrap – for responsive design (optional)
* HTML5 + CSS3 – for page structure and styling
* FontAwesome – for icons and visual elements

**Algorithm Selection**

* We used a **Time Series Forecasting Model – LSTM (Long Short-Term Memory)** due to its ability to capture temporal dependencies and nonlinear trends in sequential data like bike rentals. LSTM outperforms traditional models (e.g., ARIMA) in handling seasonality and long-term patterns.

**Data Input**

* Historical bike rental count (hourly)
* Weather conditions (temperature, humidity, wind speed)
* Temporal features (hour of the day, day of week, holiday/weekend info)
* External events or city-specific activities (if available)

**Training Process**

* Data was preprocessed to remove outliers, fill missing values, and normalize scales.
* The dataset was split into training and validation sets using **Time Series Cross-Validation**.
* Hyperparameters (epochs, learning rate, batch size) were optimized using **Grid Search**.
* Model performance was evaluated using **RMSE** and **MAE**.

**Prediction Process**

* Once trained, the LSTM model receives real-time or daily data updates to forecast hourly bike demand.
* Predictions dynamically adjust based on current weather and time factors.
* Output is served via a web-based dashboard or API for station managers and urban planners.

**Deployment**

* Model is deployed on **IBM Cloud Functions** with a REST API interface.
* User interface hosted via **IBM Cloud Object Storage (Static Site Hosting)**.
* Scheduled re-training is enabled to adapt to new data.

IBM Granite Foundation Model – https://www.ibm.com/products/granite

IBM Cloud Lite – https://www.ibm.com/cloud/free

D. R. Hines, “Audio Amplifier Design Considerations,” *Journal of Audio Engineering Society*, Vol. 49, No. 7/8, 2001.

A. S. Sedra & K. C. Smith, *Microelectronic Circuits*, Oxford University Press, 7th Ed., 2015.

S. Haykin, *Neural Networks and Learning Machines*, Pearson, 3rd Ed., 2009 – for foundational AI agent concepts.