NVIDIA Riva Multilingual Voice-Enabled RAG Chatbot

Module 3 - Lab 3

Objectives

By the end of this lab, you will:

- Build a multilingual voice-enabled RAG chatbot using NVIDIA Riva and HPE AI Essentials Knowledge Base
- Implement automatic speech recognition (ASR) with multilingual support and automatic language detection
- Integrate retrieval-augmented generation (RAG) for context-aware responses from enterprise knowledge bases
- Generate natural speech responses using text-to-speech (TTS) in multiple languages
- Understand the complete architecture of a production-ready voice AI application

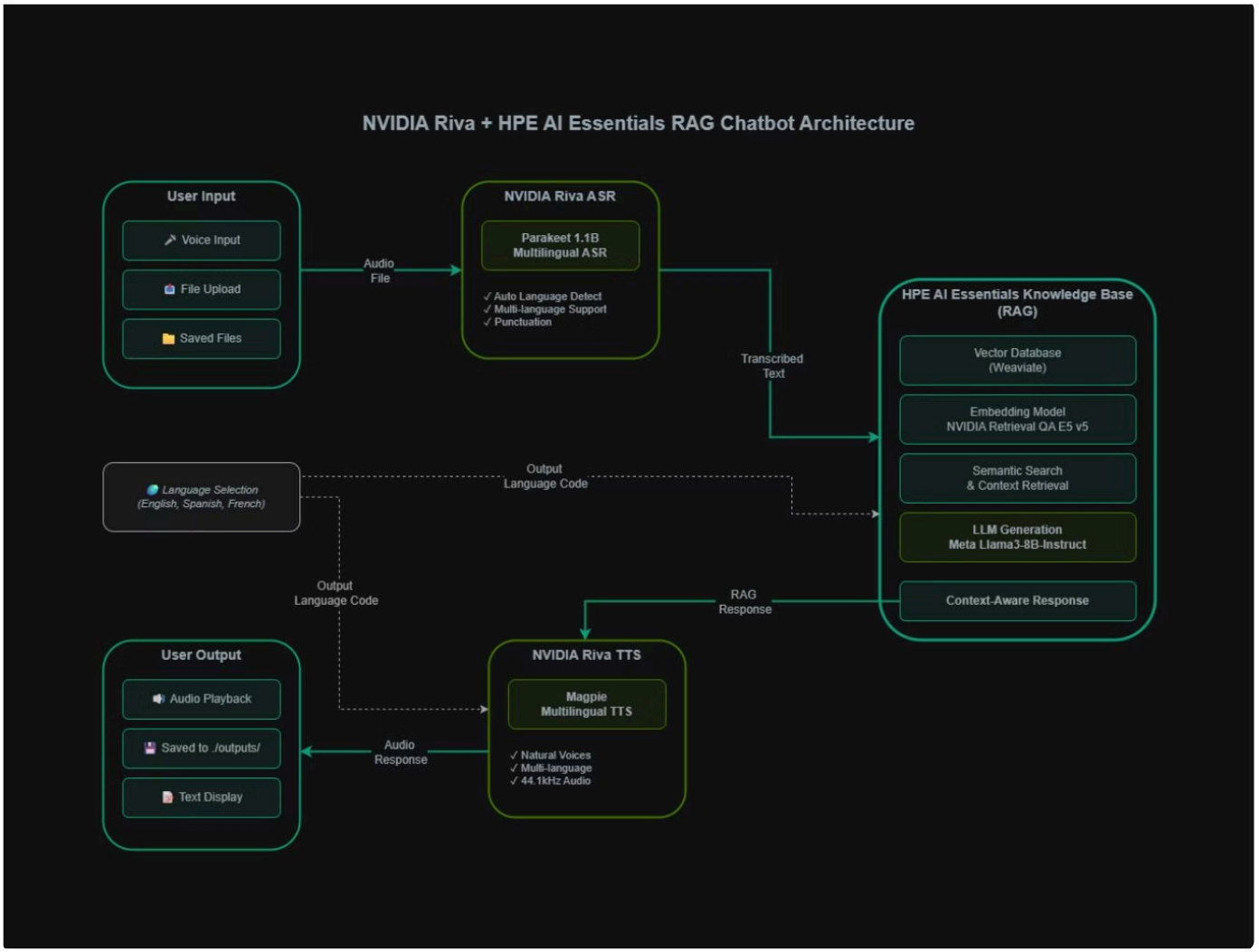
Description

In this hands-on lab, you build a complete multilingual voice-enabled chatbot that combines NVIDIA Riva's speech AI capabilities with HPE AI Essentials Knowledge Base. You start by configuring the NVIDIA Riva ASR service to transcribe user questions with automatic language detection. Next, you integrate the HPE AI Essentials RAG endpoint to retrieve context-aware answers from your enterprise knowledge base. Finally, you implement NVIDIA Riva TTS to convert text responses into natural-sounding speech in the user's preferred language. The lab demonstrates a real-world enterprise AI application where users interact with knowledge bases using voice in their native language.

Key Concepts

This lab introduces you to multimodal AI systems that combine speech recognition, natural language understanding, and speech synthesis. You work with NVIDIA Riva's Parakeet multilingual ASR model for automatic language detection, integrate with vector databases through RAG for semantic search and context retrieval, and use the Magpie multilingual TTS model for natural speech generation. The architecture demonstrates how modern AI applications orchestrate multiple specialized models to create seamless user experiences across different languages and modalities.

Architecture Overview



The system architecture consists of five main components:

01

User Input Layer

Accepts audio through pre-recorded files in the ./input/ directory

02

NVIDIA Riva ASR

Transcribes speech to text with automatic language detection using the Parakeet 1.1B multilingual model

03

HPE AI Essentials Knowledge Base

Performs semantic search on vector embeddings and generates context-aware responses using Meta Llama3-8B-Instruct

04

NVIDIA Riva TTS

Synthesizes natural speech from text responses using the Magpie multilingual model

05

User Output Layer

Delivers audio playback and saves responses to ./outputs/ directory

Task 0: Lab Setup and Environment Access

Step 1: Access the Lab Environment

Open your web browser and navigate to **http://salesdemos.ext.hpe.com/**. Sign in using your HPE Employee or Partner credentials. Enroll in the event using the event code provided by your instructor. Access the console to enter the VDI system.

Step 2: Authenticate to HPE GreenLake

Once inside the VDI, you see the GreenLake authentication page with pre-filled credentials. Click **Sign In** to authenticate and reach the HPE AI Essentials page.

Step 3: Open Jupyter Notebooks

On the HPE AI Essentials page, use the left-side navigation panel to locate and click on **Notebooks**. You see a list of available notebooks assigned to different student numbers.

Step 4: Access Your Assigned Notebook

Locate the notebook corresponding to your student number (provided by your instructor). Click on it to open your dedicated Jupyter environment.

Step 5: Navigate to Lab 3

In the Jupyter file browser, navigate to **Module 3** → **Lab 3** → **NVIDIA_Riva_RAG_Chatbot**. Open the notebook file **Riva_RAG_Chatbot_Lab.ipynb**.

You are now ready to begin the lab exercises.

Task 1: Install Dependencies and Configure Endpoints

In this task, you install the required Python libraries and configure the connection endpoints for NVIDIA Riva and HPE AI Essentials Knowledge Base.

Step 1: Install NVIDIA Riva Client Library

Execute the first code cell to install the NVIDIA Riva client library and other dependencies.

Code Cell 1:

```
!pip install nvidia-riva-client requests
```

This command installs the nvidia-riva-client library, which provides Python bindings for NVIDIA Riva's ASR and TTS services, and the requests library for making HTTP calls to the RAG endpoint.

Expected Output:

```
Successfully installed nvidia-riva-client-2.19.0 requests-2.31.0
```

Step 2: Import Required Libraries

Execute the next cell to import all necessary Python modules.

Code Cell 2:

```
import os
import wave
import json
import requests
import riva.client
import IPython.display as ipd
from IPython.display import display
```

You import standard libraries for file operations (os, wave), data handling (json), HTTP requests (requests), the Riva client SDK (riva.client), and Jupyter display utilities (IPython.display) for audio playback.

Step 3: Configure Service Endpoints

Execute the configuration cell to set up connection parameters.

Code Cell 3:

```
# Riva Configuration
RIVA_URI = "10.179.253.43:32222"
# RAG Configuration
RAG_ENDPOINT = "https://rag-coordinator.pcai2.genai2.hou"
RAG_API_PATH = "/v1/chat/completions"
AUTH_TOKEN = "YOUR_AUTH_TOKEN_HERE" # Replace with your actual token
APP_NAME = "APP_NAME_HERE" # Replace with your actual APP Name
ENABLE_CITATIONS = "false"
# Models
ASR_MODEL = "nvidia/parakeet-1.1b-rnnt-multilingual-asr"
TTS_MODEL = "nvidia/magpie-tts-multilingual"
RAG_MODEL = "meta/llama3-8b-instruct"
print(f"Riva URI: {RIVA_URI}")
print(f"RAG Endpoint: {RAG_ENDPOINT}{RAG_API_PATH}")
```

You configure the NVIDIA Riva server URI for ASR and TTS services, the HPE AI Essentials RAG endpoint for knowledge base queries, and specify the AI models you will use. The AUTH_TOKEN authenticates your requests to the RAG service.

Important: Your instructor will provide the actual AUTH_TOKEN and APP_NAME value. Replace YOUR_AUTH_TOKEN_HERE and APP_NAME_HERE with the values provided.

Expected Output:

```
Riva URI: 10.179.253.43:32222
RAG Endpoint: https://rag-coordinator.pcai2.genai2.hou/v1/chat/completions
```

Step 4: Define System Context for RAG

Execute the cell to define the system prompt that guides the RAG model's behavior.

Code Cell 4:

```
# System Context for RAG
SYSTEM_CONTEXT = """- You are an AI assistant who analyzes the input query and
provides responses strictly based on the retrieved context and previous chat
conversations.
- If no relevant context or prior chat conversations is available, do not respond
and instead state: "I don't have enough information to answer the question."
- You must not rely on general knowledge or any data from your training. Use only
the specific details from the given context and prior chat conversations.
- If no relevant context is retrieved, do not attempt to generate a response.
- Your answers must remain strictly within the boundaries of the available context
and previous chat conversations.
- If the user sends a greeting or a polite remark, reply briefly in a friendly
manner
- Do not include greetings in responses to non-greeting queries.
- Don't assume or hallucinate when responding
- Do not include the phrase 'Based on the provided context and previous chat
conversation' in the responses.
Context:
{context}
Chat_Conversations:
{chat_history}
Query:
{query}"""
```

This system prompt instructs the LLM to generate responses based only on retrieved context from the knowledge base, preventing hallucinations and ensuring factual accuracy. The placeholders {context}, {chat_history}, and {query} are populated by the RAG coordinator service.

Task 2: Initialize NVIDIA Riva Services

In this task, you establish connections to the NVIDIA Riva server and initialize the ASR and TTS services.

Step 1: Connect to Riva Server

Execute the cell to initialize the Riva client.

Code Cell 5:

```
try:
 auth = riva.client.Auth(uri=RIVA_URI)
 asr_service = riva.client.ASRService(auth)
 tts_service = riva.client.SpeechSynthesisService(auth)
 print("✓ Riva services initialized successfully.")
except Exception as e:
 print(f"  Error initializing Riva services: {e}")
```

You create an authentication object using the Riva server URI, then initialize both the ASR service for speech recognition and the TTS service for speech synthesis. The try-except block handles any connection errors gracefully.

Expected Output:

```
✓ Riva services initialized successfully.
```

Task 3: Implement Speech Recognition Function

In this task, you create a function that transcribes audio files using NVIDIA Riva's multilingual ASR with automatic language detection.

Step 1: Define the Transcription Function

Execute the cell to create the transcribe_audio function.

Code Cell 6:

```
def transcribe_audio(audio_file):
 """Transcribe audio using Riva ASR with automatic language detection."""
 try:
 with open(audio_file, 'rb') as fh:
 data = fh.read()
 config = riva.client.RecognitionConfig(
 encoding=riva.client.AudioEncoding.LINEAR_PCM,
 max_alternatives=1,
 enable_automatic_punctuation=True,
 verbatim_transcripts=False,
 language_code="multi" # Auto-detect language
 )
 # Helper to set sample_rate_hz and audio_channel_count from the WAV file header
 riva.client.add_audio_file_specs_to_config(config, audio_file)
 response = asr_service.offline_recognize(data, config)
 if len(response.results) > 0:
 transcript = response.results[0].alternatives[0].transcript
 return transcript
 else:
 return ""
 except Exception as e:
 print(f"ASR Error: {e}")
 return ""
```

This function reads an audio file, configures the ASR service with automatic language detection (language_code="multi"), enables automatic punctuation for readability, and uses the offline_recognize method to transcribe the entire audio file. The add_audio_file_specs_to_config helper automatically extracts audio parameters like sample rate and channel count from the WAV file header.

Key Configuration Parameters:

- encoding=LINEAR_PCM: Specifies uncompressed PCM audio format
- enable_automatic_punctuation=True: Adds punctuation to improve readability
- language_code="multi": Enables automatic language detection across supported languages
- max_alternatives=1: Returns only the most likely transcription

Task 4: Implement RAG Query Function

In this task, you create a function that sends transcribed text to the HPE AI Essentials Knowledge Base and retrieves context-aware responses.

Step 1: Define the RAG Query Function

Execute the cell to create the query_rag function.

Code Cell 7:

```
def query_rag(text, output_language="English"):
 """Query RAG endpoint with language-specific instruction."""
 url = f"{RAG_ENDPOINT}{RAG_API_PATH}"
 headers = {
 "Content-Type": "application/json",
 "Authorization": f"Bearer {AUTH_TOKEN}",
 "X-SA-NAME": APP_NAME,
 "X-ENABLE-CITATIONS": ENABLE_CITATIONS
 }
 # Add language instruction to the system context
 language_instruction = f"\n\nIMPORTANT: Provide your response in {output_language} language."
 system_context_with_language = SYSTEM_CONTEXT + language_instruction
 # Payload for Chat Completion
 payload = {
 "model": RAG_MODEL,
 "messages": [
 {"role": "system", "content": system_context_with_language},
 {"role": "user", "content": text},
 ],
 "max_tokens": 2048,
 "temperature": 0,
 "frequency_penalty": 0,
 "presence_penalty": 1,
 "top_p": 0.5
 }
```

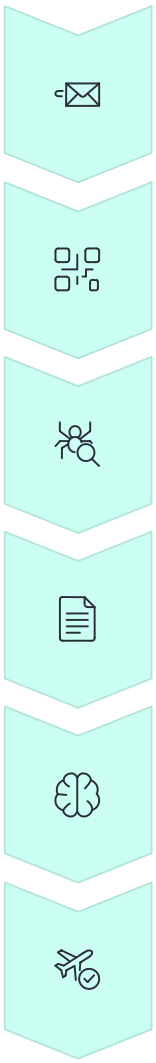
```
print(f"Querying Knowledge Base... (Requesting {output_language} response)")
try:
response = requests.post(url, headers=headers, json=payload, verify=False)
response.raise_for_status()
result = response.json()
# Handle Chat Completion Response
if "choices" in result and len(result["choices"]) > 0:
answer = result["choices"][0]["message"]["content"]
return answer
else:
return f"Unexpected response format: {str(result)}"
except Exception as e:
print(f"RAG Error: {e}")
return "I'm sorry, I couldn't connect to the knowledge base."
```

This function constructs an HTTP POST request to the RAG endpoint with the user's query and system context. You append a language instruction to ensure the LLM generates responses in the specified output language. The function uses the OpenAI-compatible chat completion API format with the system role containing the context template and the user role containing the actual query.

Key Parameters:

- temperature=0: Ensures deterministic, factual responses
- max_tokens=2048: Limits response length
- presence_penalty=1: Encourages diverse vocabulary
- top_p=0.5: Uses nucleus sampling for response generation

RAG Workflow



RAG coordinator receives your query

Generates embeddings using NVIDIA Retrieval QA E5 v5

Performs semantic search in Weaviate vector database

Retrieves relevant context chunks

Sends context + query to Llama3-8B-Instruct

LLM generates response based only on retrieved context

Task 5: Implement Text-to-Speech Function

In this task, you create a function that converts text responses into natural speech using NVIDIA Riva TTS.

Step 1: Create Output Directory

Execute the cell to set up the output directory for TTS audio files.

Code Cell 8:

```
# Counter for unique filenames
tts_counter = 0
OUTPUT_DIR = "./outputs"
os.makedirs(OUTPUT_DIR, exist_ok=True)
```

You create an outputs directory to store generated TTS audio files and initialize a counter for unique filenames. The exist_ok=True parameter prevents errors if the directory already exists.

Step 2: Define the TTS Function

Execute the cell to create the speak_text function.

Code Cell 9:

```
def speak_text(text, language_code="en-US"):
 """Synthesize speech in the specified language."""
 global tts_counter
 try:
 print(f"Synthesizing speech: {text[:50]}...")
 sample_rate_hz = 44100
 nchannels = 1
 sampwidth = 2
 response = tts_service.synthesize(
 text,
 language_code=language_code,
 sample_rate_hz=sample_rate_hz
 )
```

```
# Save to output folder
tts_counter += 1
output_file = os.path.join(OUTPUT_DIR, f"response_{tts_counter}.wav")
with wave.open(output_file, 'wb') as out_f:
out_f.setnchannels(nchannels)
out_f.setsampwidth(sampwidth)
out_f.setframerate(sample_rate_hz)
out_f.writeframesraw(response.audio)
print(f"✓ Audio saved to: {output_file}")
display(ipd.Audio(output_file, autoplay=True))
except Exception as e:
print(f"TTS Error: {e}")
```

This function calls the Riva TTS service to synthesize speech from text in the specified language. It configures the audio output as 44.1kHz mono WAV format, saves the generated audio to the outputs directory with a unique filename, and displays an audio player in the notebook with autoplay enabled.

Audio Specifications:

Sample rate: 44,100 Hz (CD quality)

Channels: 1 (Mono) Bit depth: 16-bit

Format: WAV (uncompressed PCM)

Supported Languages:

English (en-US)

Spanish (es-US)

French (fr-FR)

Task 6: Launch the Interactive Chatbot UI

In this task, you launch the interactive user interface that orchestrates the complete voice interaction workflow.

Step 1: Import and Launch the UI

Execute the final cell to start the chatbot interface.

Code Cell 10:

```
from chatbot_ui import create_chatbot_ui
# Launch the chatbot UI
create_chatbot_ui(transcribe_audio, query_rag, speak_text)
```

You import the create_chatbot_ui function from the chatbot_ui.py module and pass the three functions you created (ASR, RAG, TTS) as parameters. The UI provides a dropdown to select the output language and a file selector to choose pre-recorded audio questions from the ./input/ directory.

Expected Output: You see an interactive widget interface with multiple input options and controls.

Step 2: Understanding the User Interface

The chatbot UI provides a complete interface for interacting with the voice-enabled RAG system. Let's explore each component:

UI Components Explained:

Note Banner

- Displays important information: "TTS supports English, Spanish, and French only"
- Reminds you of the available output languages

Output Language Dropdown

- Select your preferred response language
- Options: English (US), Spanish (US), French
- Controls both RAG response generation and TTS synthesis
- The RAG system will generate answers in this language
- The TTS will speak the response in this language

Record Audio Section (Disabled in VDI)

- "Start Recording" and "Stop Recording" buttons
- Note: These buttons are not functional in the VDI environment due to microphone access restrictions
- In environments with microphone access, you can record questions directly

Upload Audio File

- "Upload Audio" button allows you to upload audio files from your local system
- Accepts .wav and .mp3 formats
- Uploaded files are automatically saved to the ./input/ directory
- Useful for testing with your own audio recordings

Select Saved File Section

- **Saved Files** dropdown: Lists all audio files in the ./input/ directory
- **Refresh** button: Updates the file list after adding new files
- **Process Selected** button: Starts the complete ASR → RAG → TTS workflow

This is the primary method for the lab since microphone access is restricted

Output Area

- Displays real-time progress messages
- Shows transcribed text from ASR
- Displays RAG response text
- Contains audio player for TTS output

Step 3: Example Interaction

Here's what happens when you process an audio file:

Step 4: Select an Audio File

Use the **Saved Files** dropdown to select one of the pre-recorded question files from the ./input/ directory. Your instructor has provided sample questions in multiple languages.

Step 3: Choose Output Language

Select your preferred output language from the **Output Language** dropdown. The RAG response will be generated in this language, and the TTS will speak in this language.

Step 4: Process the Query

Click the **Process Selected** button to execute the complete workflow

Step 5: Try Different Languages

Experiment with different combinations:

- Select an English question → Output in Spanish
- Select a Spanish question → Output in French
- Select a French question → Output in English

Observe how the ASR automatically detects the input language while the RAG and TTS use your selected output language.

Workflow Demonstration:

1. File Selection

1

- Selected file: input.wav from the dropdown
- Clicked "Process Selected" button

2. Processing Messages

2

- "Processing file: input.wav" Confirms file selection
- "Transcribing..." ASR is processing the audio

3. ASR Output

3

- "You asked: What is hpe ai essentials" Displays the transcribed question
- Note: ASR automatically detected English language

4. RAG Processing

4

- "Querying Knowledge Base (requesting English response)..." Shows RAG is working
- "Querying Knowledge Base... (Requesting English response)" Confirms language selection

5. RAG Response

5

"Bot Answer: HPE AI Essentials Software provides a suite of proprietary and open-source tools for building generative AI applications, such as chatbots, with low-code features and automated accelerators."

The response is generated from the knowledge base context

6. TTS Synthesis

6

- "Speaking answer in English..." TTS is generating speech
- "Synthesizing speech: HPE AI Essentials Software provides a pro..."
- Shows first 50 characters of the text being synthesized

7. Audio Output

7

- "✓ Audio saved to: ./outputs/response_1.wav" Confirms file saved
- Audio player appears with playback controls
- Duration shown: 0:11 (11 seconds)
- Audio automatically plays in the notebook

Key Observations:

- The entire workflow completes in seconds
- You see each stage of processing clearly
- The audio response is both played and saved for later review
- All interactions are logged in the output area

Troubleshooting Guide

Issue: "Error initializing Riva services"

Cause: Cannot connect to the NVIDIA Riva server.

Solution:

1. Verify the RIVA_URI is correct: 10.179.253.43:32222
2. Check network connectivity from the VDI to the Riva server
3. Ensure the Riva services are running (contact your instructor)

Issue: "RAG Error: 401 Unauthorized"

Cause: Invalid or missing authentication token.

Solution:

1. Verify you replaced YOUR_AUTH_TOKEN_HERE with the actual token
2. Ensure the token is correctly copied without extra spaces
3. Request a new token from your instructor if needed

Issue: "TTS Error: Voice not found for language"

Cause: Selected a language not supported by the deployed TTS model.

Solution:

1. Use only the supported languages: English (en-US), Spanish (es-US), or French (fr-FR)
2. The Magpie multilingual model deployed on this server supports only these three languages

Issue: "ASR returns empty transcription"

Cause: Audio file format incompatibility or corrupted file.

Solution:

1. Verify the audio file is in WAV format, 16kHz, mono
2. Check the file is not corrupted (try playing it locally)
3. Ensure the audio contains clear speech (not silence or noise)

Summary

In this lab, you successfully built a complete multilingual voice-enabled RAG chatbot by integrating NVIDIA Riva's speech AI capabilities with HPE AI Essentials Knowledge Base. You configured the Parakeet multilingual ASR model to automatically detect and transcribe speech in multiple languages, integrated with the RAG endpoint to retrieve context-aware answers from enterprise knowledge bases using semantic search, and implemented the Magpie multilingual TTS model to generate natural speech responses in the user's preferred language.

You learned how modern AI applications orchestrate multiple specialized models to create seamless multimodal experiences. The ASR service handles speech-to-text conversion with automatic language detection, the RAG system performs vector similarity search and context retrieval from Weaviate, and the TTS service converts text back to natural speech. This architecture demonstrates production-ready patterns for building enterprise voice AI applications that can serve users across different languages and modalities.

Key takeaways include understanding how vector embeddings enable semantic search, how RAG prevents LLM hallucinations by grounding responses in retrieved context, and how multilingual models can automatically adapt to different languages without explicit configuration. You also gained hands-on experience with NVIDIA Riva's gRPC APIs and HPE AI Essentials' OpenAI-compatible chat completion endpoints.

Additional References

NVIDIA Riva Documentation

- **Riva User Guide**: https://docs.nvidia.com/deeplearning/riva/user-guide/docs/
- **Riva ASR API Reference**: https://docs.nvidia.com/deeplearning/riva/user-guide/docs/asr/asroverview.html
- **Riva TTS API Reference**: https://docs.nvidia.com/deeplearning/riva/user-guide/docs/tts/ttsoverview.html
- **Parakeet ASR Model**: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/riva/models/parakeet-rnnt-1.1b
- **Magpie TTS Model**: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/riva/models/speechsynthesis_multilingual_magpietts_ipa

HPE AI Essentials

- **AI Essentials Documentation**: https://support.hpe.com/hpesc/public/docDisplay?docId=a00120892en_us
- **RAG Architecture Guide**: https://www.hpe.com/us/en/ai-essentials.html
- **Vector Database (Weaviate)**: https://weaviate.io/developers/weaviate

Retrieval-Augmented Generation

- **RAG Paper**: https://arxiv.org/abs/2005.11401
- **Vector Embeddings Guide**: https://www.pinecone.io/learn/vector-embeddings/
- **Semantic Search Explained**: https://www.elastic.co/what-is/semantic-search

Python Libraries

- **nvidia-riva-client**: https://pypi.org/project/nvidia-riva-client/
- **Requests Library**: https://requests.readthedocs.io/
- **IPython Display**: https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html

Additional Learning

- **NVIDIA Deep Learning Institute**: https://www.nvidia.com/en-us/training/
- **Building RAG Applications**: https://developer.nvidia.com/blog/building-rag-applications/
- **Multimodal AI Systems**: https://developer.nvidia.com/blog/multimodal-ai/