A module/product solution provider

JJSDK_ VPS_RAG Extension User Guide

Version: 1.0.0

Contact: Brian Huang (brianbaby0409@gmail.com)

CHANGE HISTORY

Version	Date	Description
V1.0.0	2025.8.12	First Complete Version for JJSDK(1.1.1) Extension

CATALOGUE

1 Dependency

1.1 Packages

- **GLTFUtility** - Allows you to import and export **glTF/glb** files (like your map data from Immersal) during runtime and in editor.
- **Newtonsoft Json** - LINQ to JSON for manually reading and writing JSON. Commonly used in request/response format translation.
- **Immersal SDK** - The Immersal SDK is a spatial mapping & visual positioning system. In this case, we use the SDK to import our map and edit the digital content in the editor.
- **WavUtility** - For saving and loading wav files in Unity.
- **JJ SDK v1.1.1** - In this case, we use this SDK to obtain camera information and device orientation for VPS & IMU-based positioning and orientation.

1.2 Compatibility

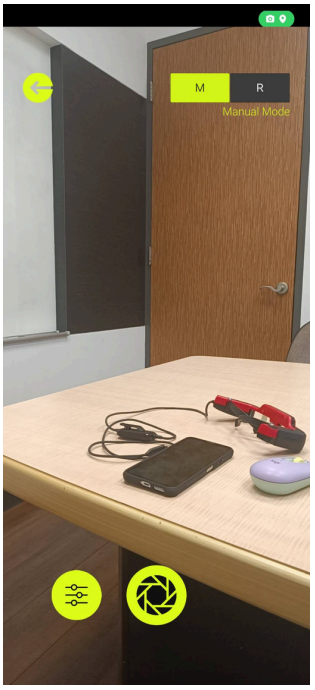
- Unity 2022.3 LTS
- AR Foundation 5.1+
- must set the project's minSdkVersion to API level 27

Note: Earlier versions of Unity and AR Foundation will still work with minimal script changes.

2 Immersal SDK Setup

2.1 Scanning - Immersal Mapper

Immersal Mapper

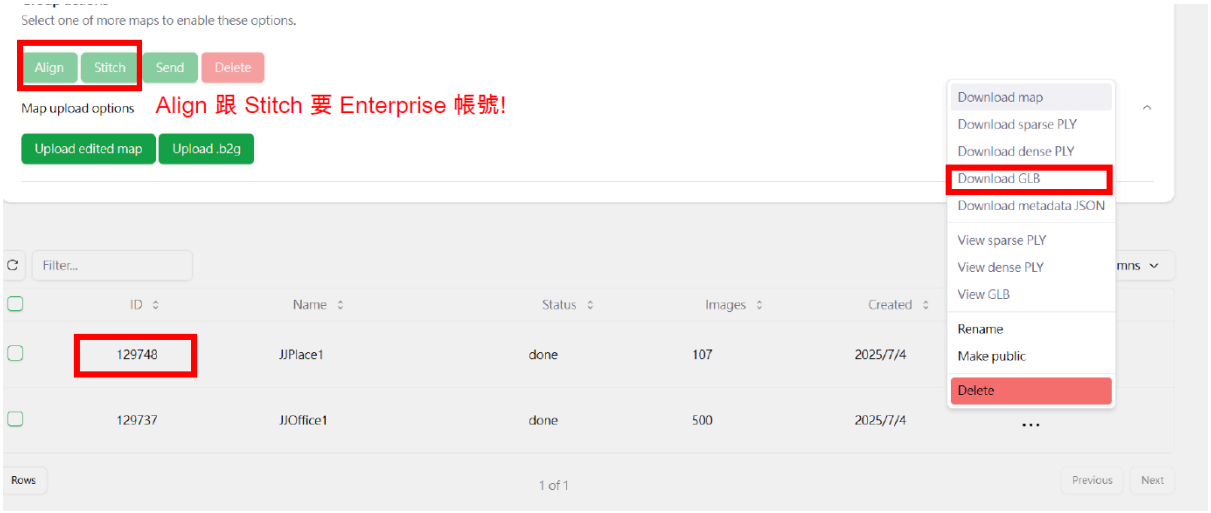


Select **Manual Mode** for scanning to generate a file with textures.

Here are some tips more scanning the indoor environment:

- Keep the camera moving slowly and steadily.
- In addition to moving horizontally, tilt the camera up or down to capture features from the tops or bottoms of objects.
- Ensure 50–70% overlap between scanning paths to avoid localization gaps.
- Avoid plain white walls, glass, or reflective surfaces, as these areas have few feature points and can easily cause localization failures.
- Avoid strong backlighting or overly bright areas.

2.2 Download Map - Immersal Portal



For the Pro Account:

- A single map can contain up to 500 photos.
- Supports importing externally edited point cloud files.
  Supports importing point cloud files captured with Polycam, 360 cameras, Matterport, or LiDAR.
- Note that the function of Align and Stitch needs the Enterprise Account.

What you need to do in this step is **download the GLB file** (which includes materials and textures) from the Immersal Portal and note down the **Map ID**.

2.3 Importing and Editing - Unity Editor

Figure 1. Project Hierarchy

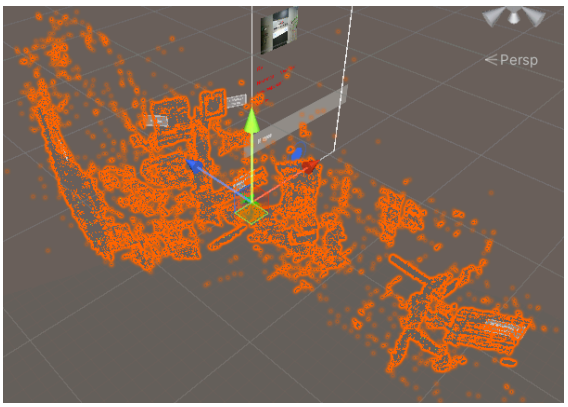


Figure 2. Point Cloud in Scene

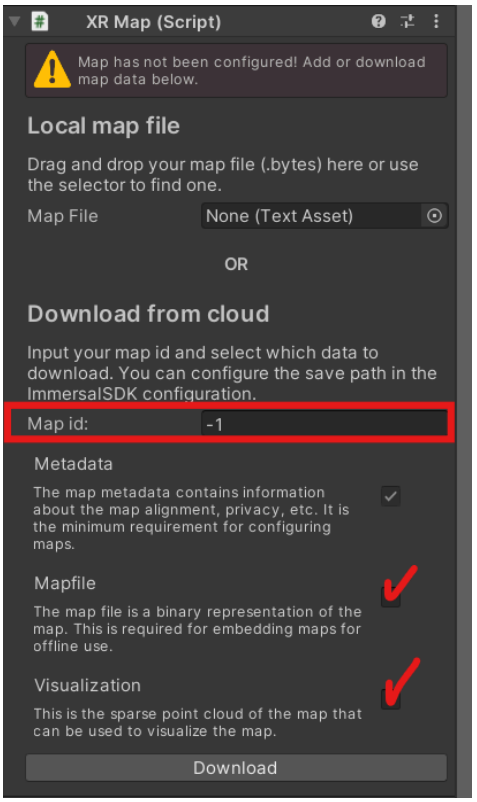


Figure 3. XR Map Inspector

First, in the sample scene, locate the gameobject **XR Space** (or create your own) and find the **XR Map** script.

Enter the **Map ID** to import the target point cloud into the scene.

Then, you should see a new point cloud object appear under the **XR Map**'s child objects, and the scene should look like **Figure 2.**

Additionally, you can place the GLB map model downloaded from the Immersal Portal into the **XR Space**'s child objects, and you will notice that it aligns correctly with the XR Map's point cloud model.

Finally, after placing your digital content in the scene at the appropriate position and orientation, you can create a **"Custom"** empty GameObject under **XR Space** and set them as children of **Custom**. (Refer to **Figure 1**.)

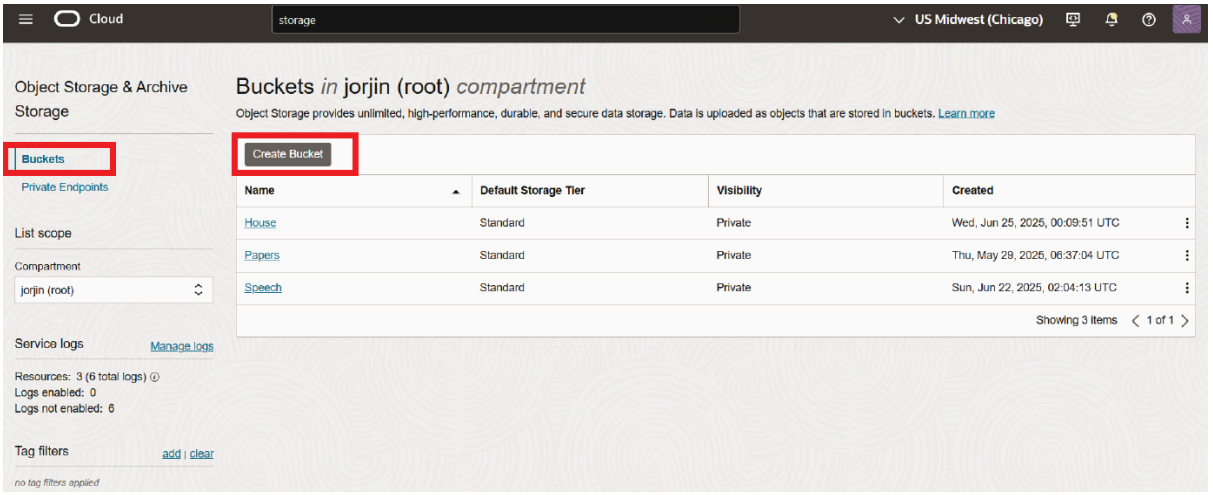
All Done!

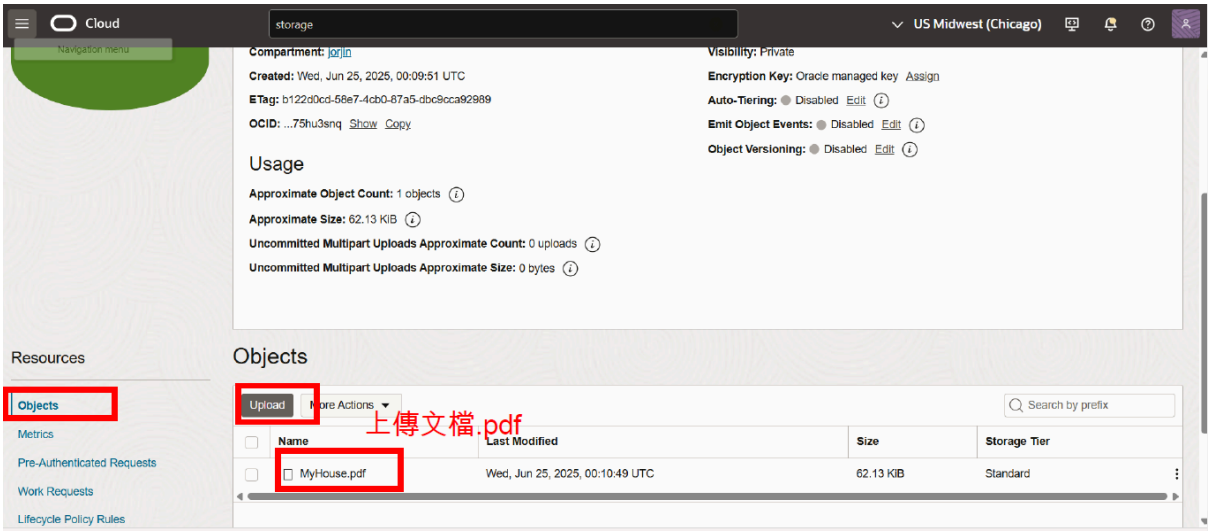
3 OCI RAG Setup

After completing the scene editing, you can use the position of the digital content and the viewing angles to create a RAG document (.pdf).

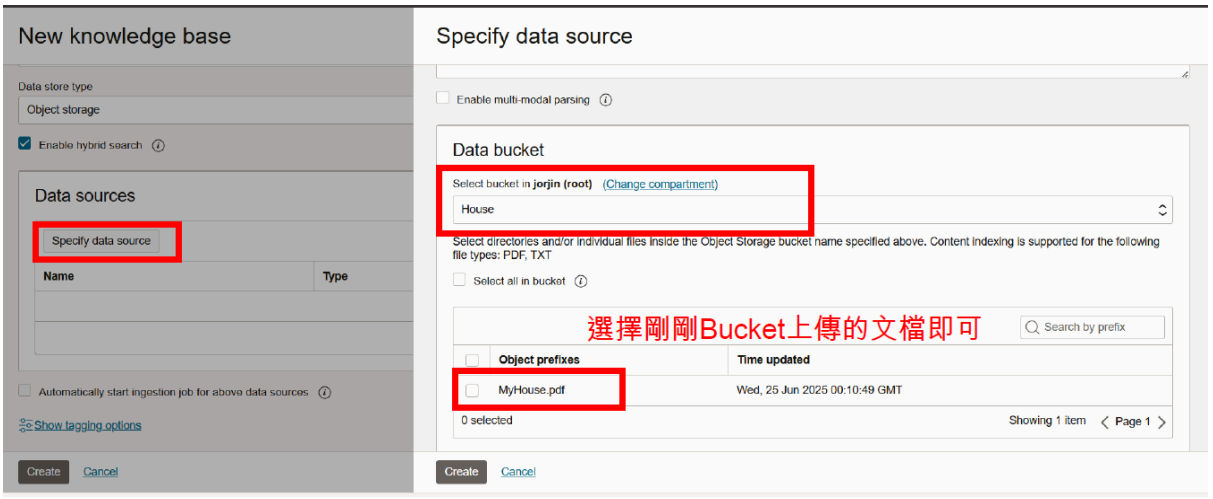
Before starting 3.1, please create your OCI account and ensure that the server region supports RAG (recommended: **Chicago**).

3.1 Upload Document - Bucket, OCI Storage

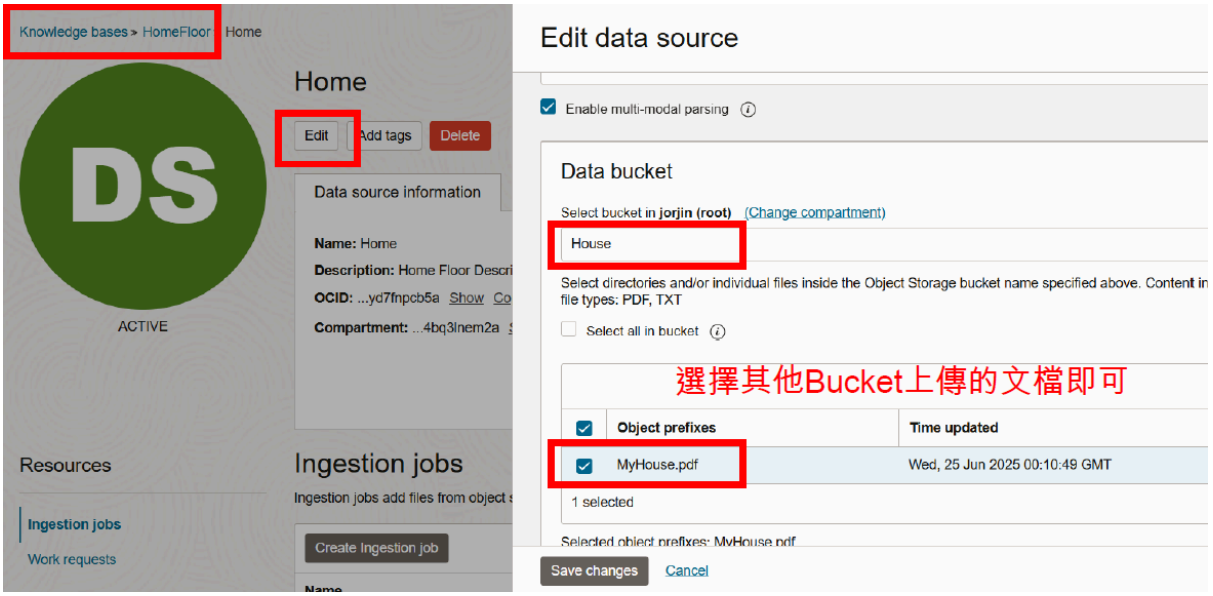




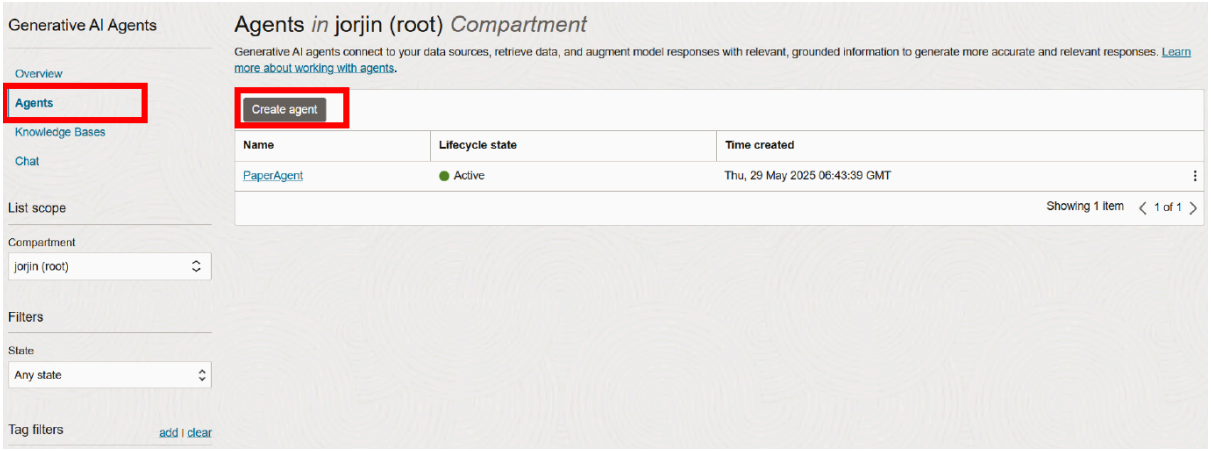
3.2 Create RAG Agent - Knowledge Bases, Generative AI Agents



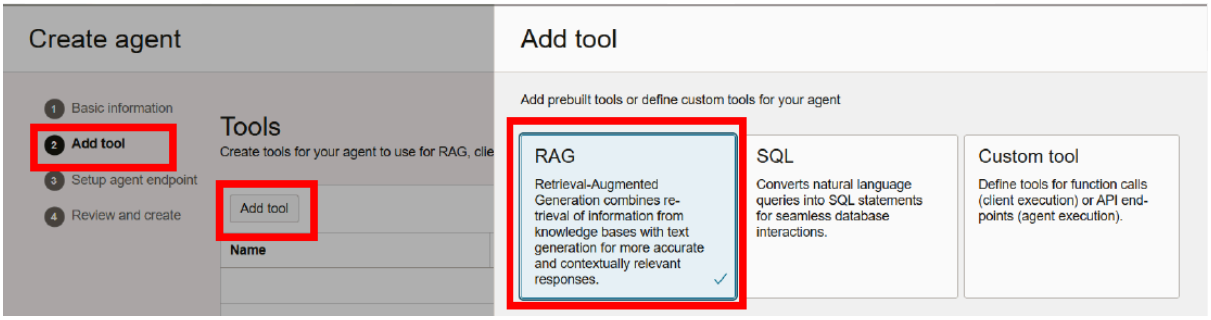
3.25 Renew Knowledge Bases

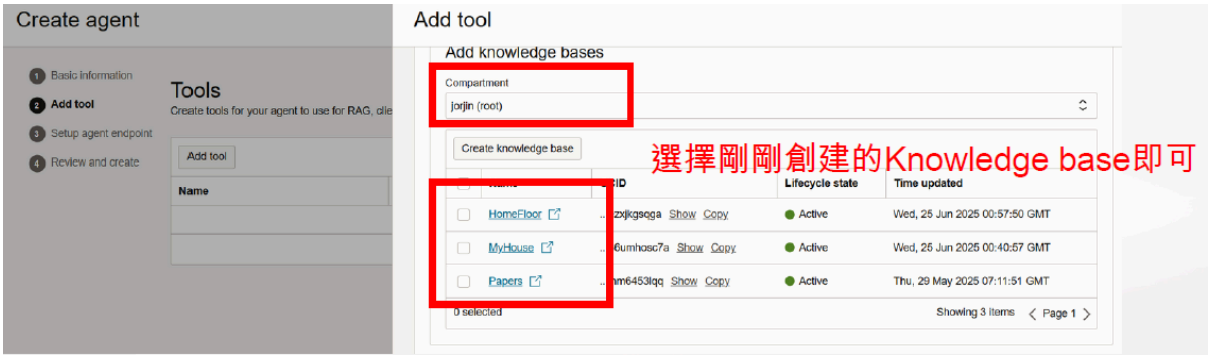


3.3 Create RAG Agent - Agents, Generative AI Agents

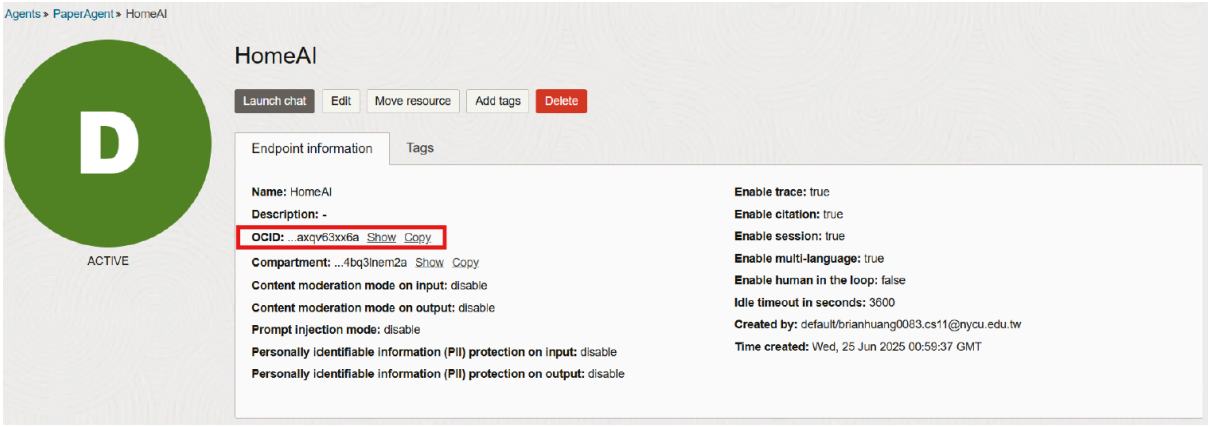


Setting your agents…

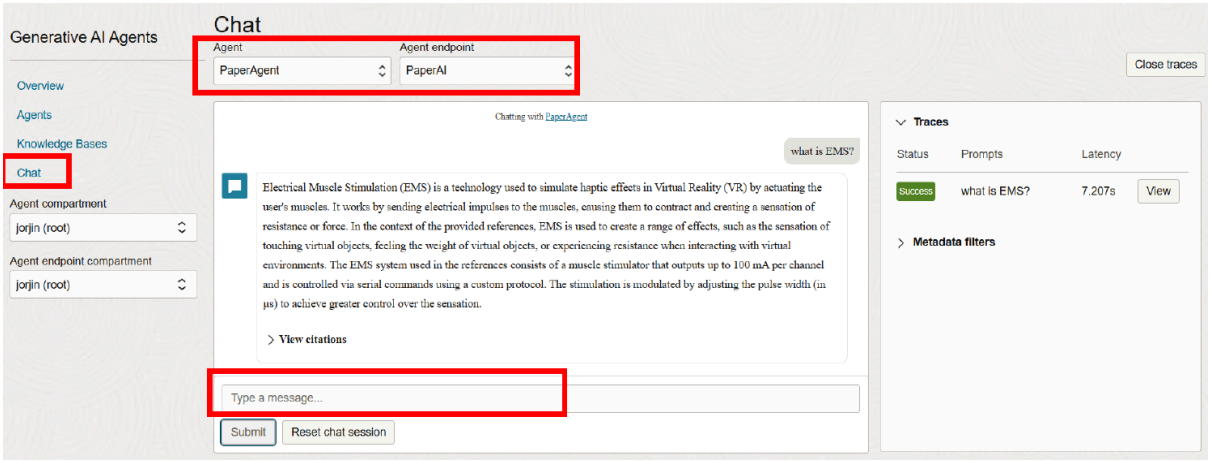




3.4 Finish and Test



After finishing the setting of your RAG agent, record the **OCID** of the agent.

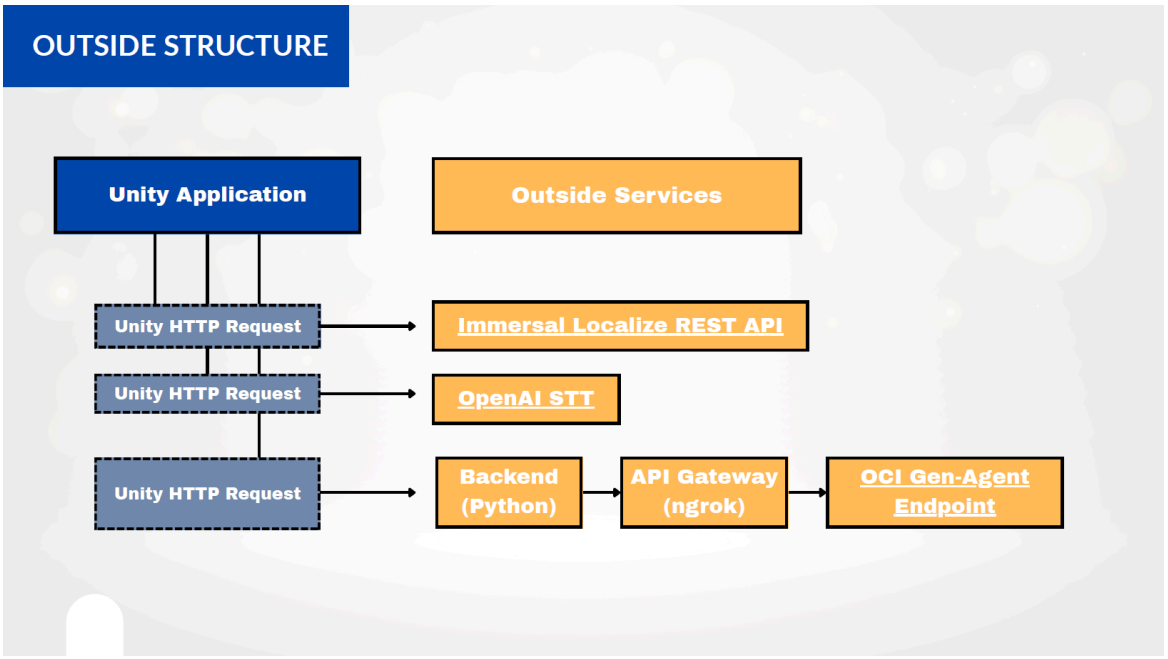


Once you have confirmed that your agent's **Endpoint Status** is set to *Active*, you can proceed to the **Chat** (Generative AI Agents>Chat) to test whether the RAG functionality is working properly.

3.5 Tutorial Resources

- Related OCI Backend Tutorial: OCI_RAG_Document_Result.pdf

4 Python Backend



In the backend section, I will introduce how to use Python and OCI to implement STT, TTS, and RAG functionalities.

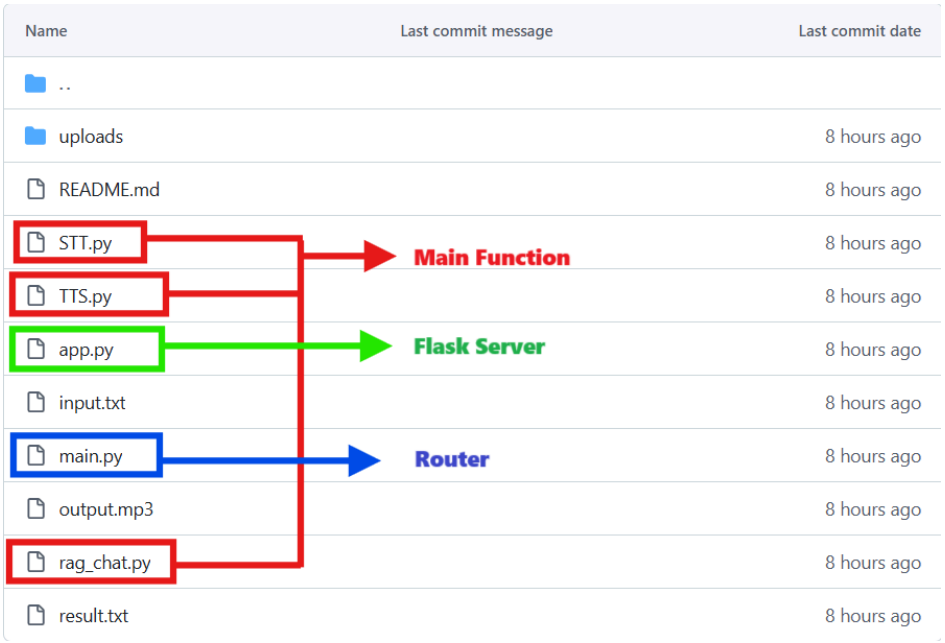


Figure. File type in the repo

4.1 How to use - Starter Note

Download Project on Github

- git clone https://github.com/BrianGodd/JJSDK_VPS_RAG_Extension
- cd JJSDK_VPS_RAG_Extension\python(Backend)

Install Environment Requirement

- pip install --upgrade pip
- pip install -r requirements.txt

Setting the user information (from OCI)

- STT.py > NAMESPACE: enter your upload **bucket namespace**
- STT.py > COMPARTMENT_ID: enter your **COMPARTMENT_ID**
- TTS.py > COMPARTMENT_ID: enter your **COMPARTMENT_ID**
- rag_chat.py > agent_endpoint_id: enter your agent **endpoint OCID**

Run the Server

- python app,py => it should open the port 5005
- open PowerShell > ngrok http 5005 (https://ngrok.com/downloads/windows) => open the Gateway for your phone app

4.2 STT - *STT.py*

Configuration

- model_type: “ORACLE” or “WHISPER_MEDIUM”
- language_code: choose the appropriate code name for the target model_type
- BUCKET_NAME: the bucket name where you upload to OCI
- input_prefix: input audio files’ folder name in the bucket
- output_prefix: result files’ folder name in the bucket

Functions

- *upload_audio_file*: upload the recording audio file(.WAV) to OCI Bucket
- *boost_upload_audio_file*: convert and upload the audio file(.FLAC) to OCI Bucket

- *create_speech_job*: create a speech transcription job
- *wait_for_job_completion*: wait for the transcription job to complete. Polls every 2 seconds in default
- *download_speech_json*: download the transcription from OCI Bucket
- *create_transcript*: extract the transcript string from JSON data
- *cleanup_all_job_output*: clear all files in the Bucket output folder

STT Flow (around 10s)

1. convert and upload the audio file(.FLAC) to OCI Bucket
2. create a speech transcription job
3. wait for the transcription job to complete
4. download the transcription JSON from OCI Bucket
5. extract the transcript string from JSON data
6. return transcript to *main.py* to execute the RAG process

4.3 TTS - *TTS.py*

Configuration

- language: only **English** available
- model_family: only **ORACLE** available
- model_name: TTS_1_STANDARD (machine sound) / TTS_2_NATURAL (Human sound)
- voice_id: Brian /Annabelle /Bob /…
- output_format: MP3 / PCM / OGG / JSON

Functions

- *Text2Speech*: Creates an audio for the given input text based on other input parameters.

The execution time depends on the length of the input text, ranging from as short as 2 seconds to as long as **4–5 seconds**.

By the way, the output audio file will be saved as “output.mp3” in the backend folder.

4.4 RAG - *rag_chat.py,* Chat & Session

Prompt

- "I’m standing at ({**pos_x**}, {**pos_z**}) which means (X, Z) and facing toward {**direction**} degrees, where 0° points north (Z+) and 90° points east (X+). You can use the relative direction to answer questions, but do not use the coordinates and the compass. Based on these, please answer the following question naturally in under {**max_token_hint**} words: {**content**}"
- **content** is the transcript text from the STT

Functions

- *get_or_create_session*: If a valid session already exists, return its ID; otherwise, create a new session, store its ID in cache, and return it.
- *load_cached_session*: Read the stored session ID from cache JSON
- *save_cached_session*: Save the session ID to the cache JSON
- *validate_session*: use *client.get_session()* to check the session’s state
- *Chat*: create the new client, call *get_or_create_session* and use *client.chat()* to get the RAG agent response.

The execution time initially requires a **warm-up** phase, after which the average execution time is approximately **5 seconds**.

By the way, the result will be saved as “result.txt” in the backend folder.

4.5 Server - *app.py*

app.run(host="0.0.0.0", port=5005) - -> the server will run at localhost:5005
(*You can change the port number if needed.)

Files

- AUDIO_FILENAME: audio file recorded from unity frontend
- INPUT_TEXT_FILENAME: STT result from unity (if choosing openai)
- OUTPUT_FILENAME: TTS result file from *TTS.py* backend
- RESULT_FILENAME: RAG result file from *rag_chat.py* backend

Process ("[gateway]/process")

1. Save the audio file recorded from unity to local folder
2. Catch the input parameters: **'param'**(process kind), **'param_x'** & **'param_z'** (camera position), **'param_rot'**(camera facing direction)
3. (If needed) Catch the input **'text'** to INPUT_TEXT_FILENAME, which is the STT result from unity
4. Call the Router(*main.py*) with the input parameters
5. Return with the RAG result & TTS audio file path

Process ("[gateway]/download/<filename>")

1. Use for sending the TTS audio file to unity frontend with *UnityWebRequestMultimedia.GetAudioClip()*

4.6 Router - *main.py*, router between Server and main function

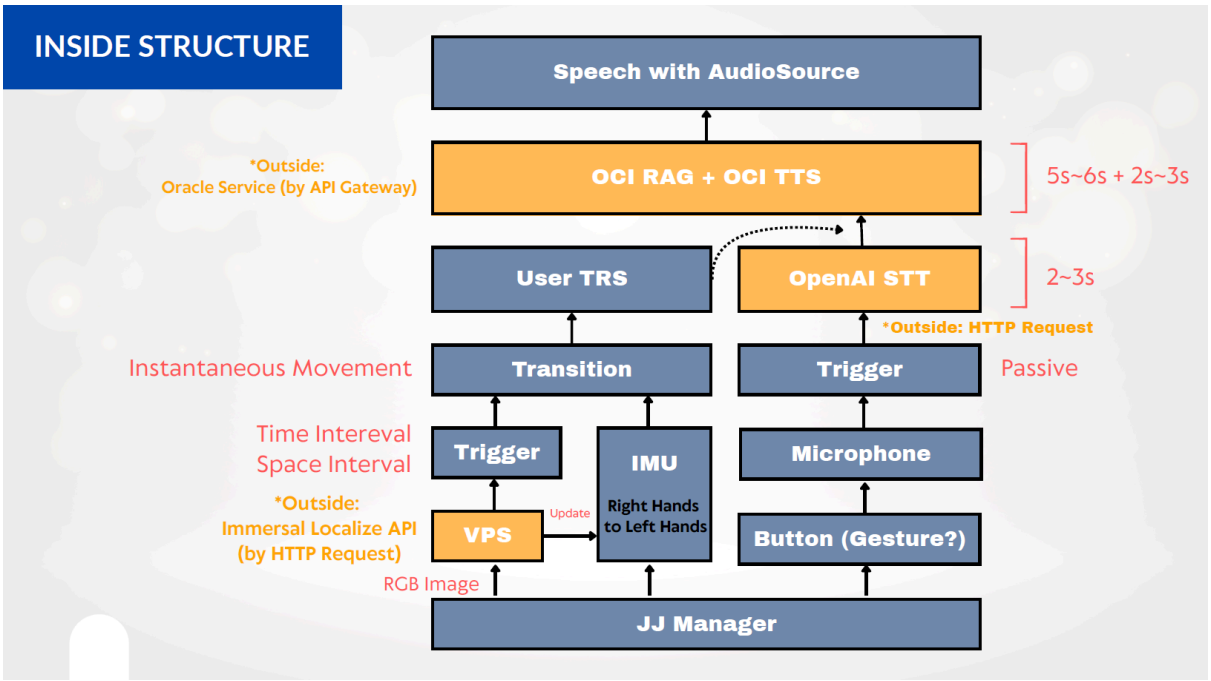
Process Kind

- **'kind' == 0:** STT & RAG & TTS use OCI service
- **'kind' == 1:** STT use OpenAI (from Unity), RAG & TTS use OCI service
- **'kind' == 2:** TTS use OpenAI (from Unity), RAG & STT use OCI service
- **'kind' == 3:** STT & TTS use OpenAI (from Unity), RAG use OCI service

Flow

1. **Start** the backend timer
2. Start **STT** service (*STT.py*) or Read the result from INPUT_TEXT_FILENAME
3. Start **RAG** service (*rag_chat.py*)
4. (if needed) Start **TTS** service (*TTS.py*)
5. **Stop** the backend timer and print out the total execution time
6. **Clean** all files in the Bucket output folder (*STT.py*)

5 Unity Frontend



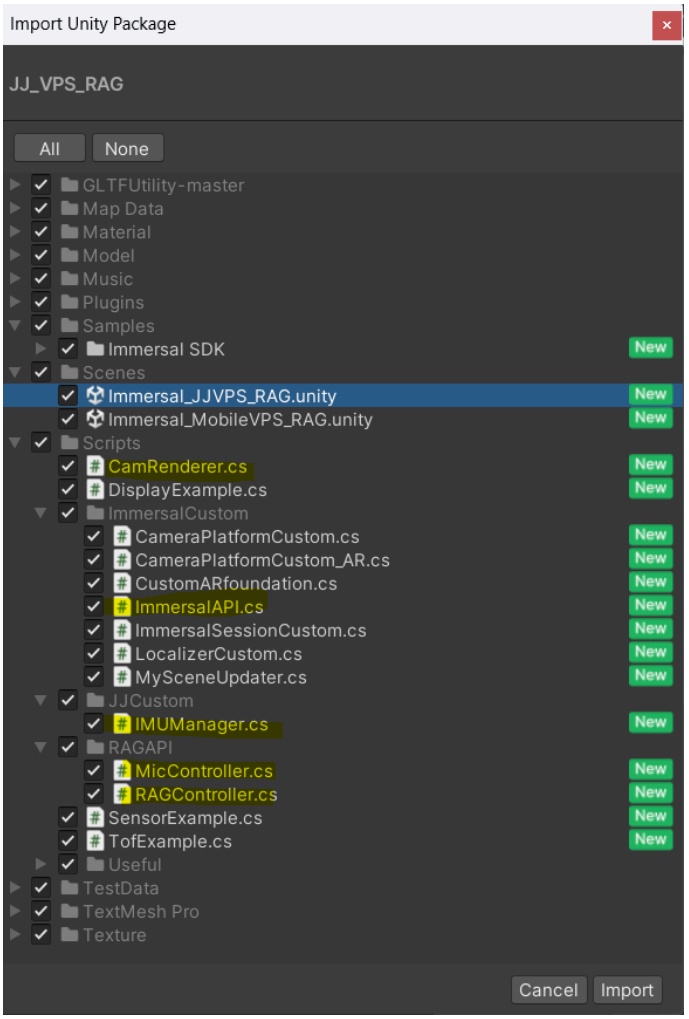
In the frontend section, I will explain how this Unity package integrates the JJSDK, Immersal API, and OCI backend to achieve spatial positioning and orientation for AR glasses, as well as AI Agent functionality.

In addition, I will provide a detailed explanation of the five main newly added scripts included in the package.

5.1 Import Unity Package

Download the package here: https://github.com/BrianGodd/JJSDK_VPS_RAG_Extension/blob/main/JJ_VPS_RAG.unitypackage

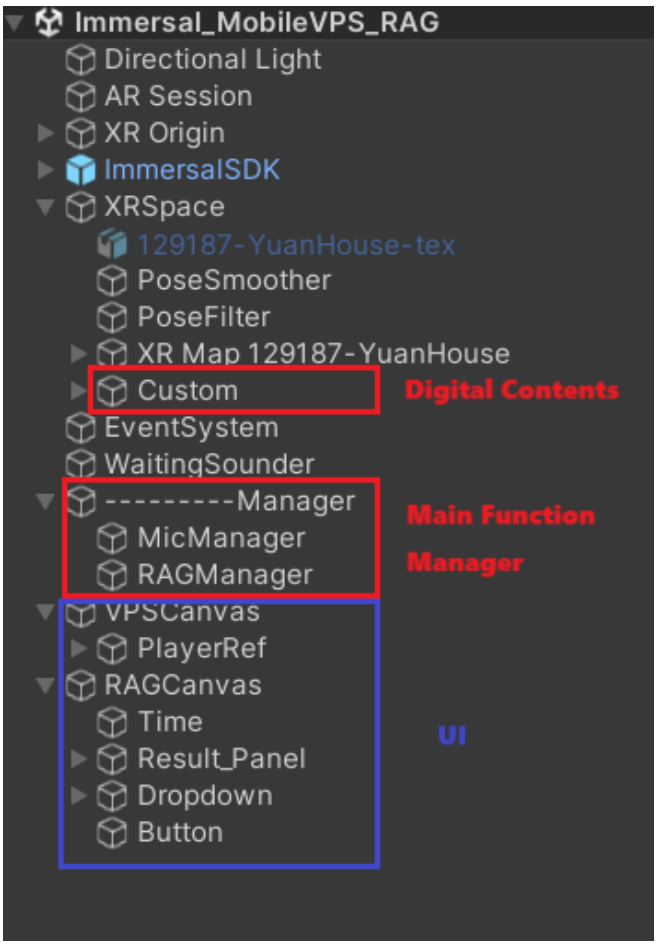
*The highlight files are the main newly added scripts which control the package functionality.



5.2 Sample Scene - Mobile ver.

Scene

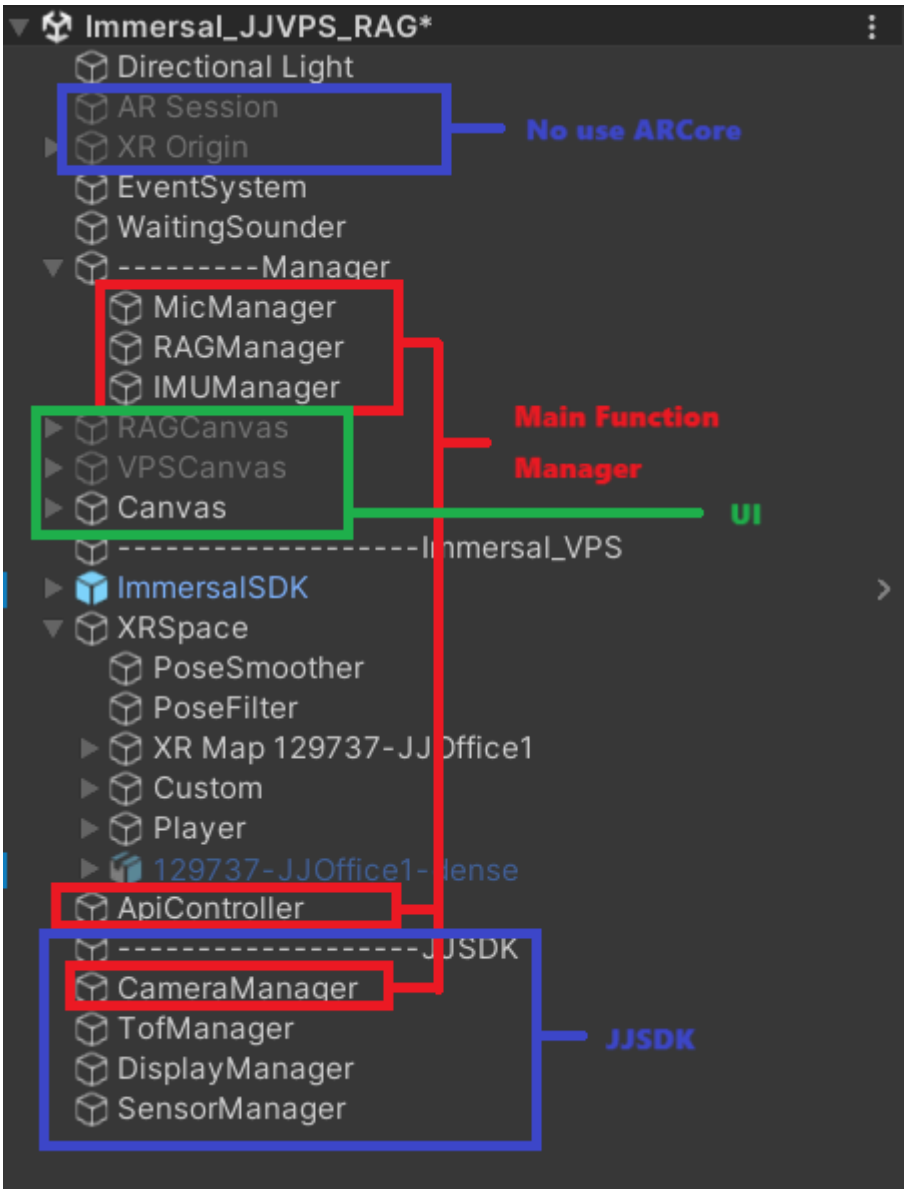
Hierarchy



5.3 Sample Scene - AR glasses ver.

Start Scene **VPS Panel** **RAG Panel**

Hierarchy



5.4 Main Function - *CameraRenderer.cs* (JJSDK)

In this script, the camera RGB array is obtained via **JJSDK**'s *onIncomingFrame()* method. *FlipTexture()* is then used to vertically flip the texture, correcting its orientation.

The corrected image data is assigned to a **RawImage** component for use in VPS localization(*ImmersalAPI.cs*).

5.5 Main Function - *ImmersalAPI.cs* (Immersal API)

Parameters

- token: Your Immersal Developer's Token
- mapIds: The map IDs you might detect

Functions

- *ConvertUIImageToBase64*: convert rawImage to base64 string type image
- *LocalizeImage*: use **HTTP**(UnityWebRequest) to POST the Immersal Localize API (*/localizeb64*) with token, mapIds, base64Image and camera intrinsics. It will then show up the localize status by the UI icon, apply the localize result to the target camera and get the map name from the response.
- *ApplyTransform*: transform the localization response to the **rotation matrix** and position vector, then **mirror them along the X-axis**. After using *transform.SetPositionAndRotation*, call *IMUManager* to **update the calibration reference**.
- *ShowLocalizeStatus*: switch the status icons by the localize status
- *GetMapName*: get the map metadata, including map name, by using **HTTP** to POST the Immersal API (*/metadataget*) with token and map id.

5.6 Main Function - *IMUManager.cs* (assist localization)

- *UpdateExternalQuaternion*: transform the Device Orientation to the valid Quaternion in Unity coordinate, then apply on the camera with the calibration reference.
- *UpdateWorldSpaceByVPS*: renew the calibration reference
- *RightHandToUnity*: keep the quaternion format, convert from a **right-handed** coordinate system to **Unity's left-handed** coordinate system (mirror the z), and then apply a 180-degree conjugate rotation around the Y-axis at the end.
- *RotateRH*: rotate 3D vector by a right-handed quaternion using quaternion-vector rotation formula. The goal is to use the known right-hand coordinate system to calculate the corresponding left-hand coordinate system.

5.7 Main Function - *MicController.cs* (Record and Save)

In this script, I implement all functionalities related to microphone recording, including the audio source dropdown menu, updating the current audio source, recording an AudioClip (start & end), and saving the AudioClip as a WAV file for use in the RAG system.

* You can find out the WavUtility.cs, which is responsible for converting an AudioClip into a byte array, on this Extension's Github Repository. (Original Source)

5.8 Main Function - *RAGController.cs* (OCI & OpenAI)

Configuration

- selectedSTT: OCI / OpenAI
- selectedTTS: OCI / OpenAI
- voice: openai tts voice, Alloy / Echo / Fable / Onyx / Nova / Shimmer
- speed: (0.25~4.0, default: 1) openai tts voice speed
- _apiKey: enter your openAI **api key**

Flow (StartRAG > UploadAudioCoroutine)

1. **Start** Unity timer
2. if 'selectedSTT' == OpenAI (kind = 1 or 3), call **STTOpenAI**
3. use HTTP(UnityWebRequest) to **communicate with the Backend** by sending the recording Wav, parameters and the STT result.
4. catch the **RAG response result** text to do the TTS process if needed.
5. if 'selectedTTS' == OpenAI (kind = 2 or 3), call **TTSOpenAI**
6. Download the **TTS audio file** from the Backend and play it
7. **Stop** Unity timer and display it on the UI text

Other Functions

- *DownloadAndPlayAudio*: use HTTP to download the target audio file by URL and play it.
- *PlayAudioFromBytes*: save the byte array audio to the target format audio file, like '.mp3'. Play it at runtime by *UnityWebRequestMultimedia.GetAudioClip()*. Finally, delete the file.
- *STTOpenAI*: use OpenAI **whisper-1** model to transcript the speech to english text.
- *TTSOpenAI*: use OpenAI **tts-1** model to do the real-time speech generation (.mp3).

5.9 Demo Video - Mobile ver.

https://www.youtube.com/watch?v=JlpbgchgMnw