# Installation Manual

## 1. Explanation for Not using Docker

This project ultimately did not adopt the Docker deployment method, mainly based on the following considerations.

### The hardware resource requirements are too high.

Our system involves local deployment of large language models (LLMs) and video generation components, and the operating environment has a very high demand for GPU resources - about 40GB of video memory. The resource requirements exceed the hardware configuration of most single GPU servers. In order to meet the development and testing needs, our customer has provided a customized development environment based on Docker, which can support the operation of the current system.

On this basis, re-packaging the Docker image will lead to complex technical issues of Docker-in-Dockers (nested Docker), including GPU mapping conflicts, driver compatibility issues, resource occupation, etc., and there is no suitable environment to support such tests at present.

Therefore, we cannot deploy a new Docker image on the server provided by the customer, nor can we build or test the image locally.

### The customer explicitly requested not to use Docker.

Our project's customer, Professor Sanjay, made it clear that he did not want us to use Docker for deployment, mainly because he wanted the system to be easily developed and integrated in the future, especially in the customer's existing infrastructure. Although Docker is convenient for packaging environments, it restricts the flexible modification of underlying components in some scientific research projects, which is not conducive to future maintenance.

### It has been officially approved by the course coordinator.

Based on the above reasons, we explained the actual situation to the course team and requested an exemption. The course coordinator, Dr. Basem Suleman, has officially approved our use of detailed installation and deployment documentation instead of Docker image deployment, and has made it clear that this method can be considered equivalent to meeting the deployment requirements.

We have provided a complete set of installation instructions in the following report and have verified the feasibility of the installation and operation process in a new environment to ensure reproducibility and stability. The following is an excerpt of Dr. Basem Suleman's original email reply:

> Dear Chengxin and the Team,
>
> Thank you for bringing this issue to my attention.
>
> In this case, it should be fine to create a detailed guide for installing/deploying the application. This will be equivalent for the Docker requirements. Please note you should explain this in your report as well as to your tutor.
>
> Please complete the **myExperience**. We would appreciate your feedback and experience from this course!
>
> Best Regards,
>
> Dr. Basem Suleiman

In summary, this project has chosen detailed installation instructions as an alternative to Docker deployment and has been approved by the course team after fully considering the technical feasibility and customer needs. We will ensure that this alternative has good usability, maintainability, and scalability in actual operation.

## 2. Installation of Front-end

### Environment requirements

- Node.js ⩾ 18 (LTS recommended)
- Npm ⩾ 9 or pnpm/yarn
- Browser:

Chrome/Edge latest version (voice recognition, WebRTC better)

### Installation steps

```
# Install dependencies
$ npm install

# Start development
$ npm start

# Production Construction
$ npm run build
```

## 3. Installation of Back-end

Since this system does not use Docker, it can be deployed by cloning the repository from GitHub.

### Clone the repository

GitHub URL:

```
$ git clone  git@github.com:unsw-cse-comp99-3900/capstone-project-25t2-9900-h16c-bread1.git
 $ cd capstone-project-25t2-9900-h16c-bread1
```

### Install dependencies

Step 1 – Create and activate the Conda environment

```
$ conda create -n bread1 python=3.10  -y
 $ conda activate bread1
```

Step 2 – Install core dependencies via Pip

```
$ pip install flask==3.1.1 \
flask-cors==6.0.1 \
flask-jwt-extended==4.7.0 \
flask-mail==0.10.0 \
flask-sqlalchemy==3.1.1 \
redis==5.0.3 \
requests==2.32.4
```

## Configure and run the application

Update the port number in run.py to your desired running port, and modify the relevant settings in config.py, such as the lifespan of each token.

```
$ python run.py
```

## Requirements file

```
# --- Core Backend Framework ---
  flask==3.1.1
  flask-cors==6.0.1
  flask-jwt-extended==4.7.0
  flask-mail==0.10.0
  flask-sqlalchemy==3.1.1
  werkzeug==3.1.3
  jinja2==3.1.6
  itsdangerous==2.2.0
  blinker==1.9.0
  markupsafe==3.0.2

  # --- Database & Cache ---
  sqlalchemy==2.0.39
  redis==5.0.3
  greenlet==3.1.1

  # --- Networking & Security ---
  requests==2.32.4
  urllib3==2.5.0
  charset-normalizer==3.4.2
  idna==3.10
  certifi==2025.6.15
  pyjwt==2.10.1
```

```
pysocks==1.7.1

# --- Packaging & Typing ---
setuptools==78.1.1
wheel==0.45.1
typing-extensions==4.14.0
zipp==3.23.0
importlib-metadata==8.7.0
```

## Environment variables & secrets

```
# Flask & Security
FLASK_ENV=production
SECRET_KEY=change_me_strong_random_string
JWT_SECRET_KEY=change_me_another_strong_random_string

# CORS
CORS_ORIGINS=http://localhost:5173

# Database
DATABASE_URL=sqlite:////data/app.db   # persisted in Docker volume

# Mail (send verification code)
MAIL_SERVER=smtp.example.com
MAIL_PORT=587
MAIL_USE_TLS=true
MAIL_USERNAME=your_smtp_user
MAIL_PASSWORD=your_smtp_password
MAIL_DEFAULT_SENDER=noreply@example.com

# Redis
REDIS_URL=redis://redis:6379/0
```

## 4. Installation of LLMs

### Prerequisites

Python 3.8+

Ollama with models:

- mistral-nemo: 12b-instruct-2407-fp16
- llama3.1: 8b-instruct-q4_K_M

Milvus API service running on `localhost:9090`

Tavily API key

## Set Environment Variables

```
~/.bashrc
export  OLLAMA_MODELS=~/workspace/models
source  ~/workspace/share/conda/etc/profile.d/conda.sh
conda activate llm-dev
export  TAVILY_API_KEY="your_tavily_api_key_here"

# Optional: Configure Milvus API (if  different from default)
export  MILVUS_API_BASE_URL="http://localhost:9090"
```

## Install dependencies

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

## Install Ollama

```
$ curl -fsSL  https://ollama.ai/install.sh | sh
```

## Start Ollama and load models

```
$ ollama serve
$ ollama pull  mistral-nemo:12b-instruct-2407-fp16
```

### Start the API Server

```
$ python rag/api_interface.py
```

## 5. Installation of RAG

### Technical Framework

- Backend Framework: Flask (HTTP API)
- Database Storage: Milvus
- Programming Language: Python3
- Deep Learning Framework: PyTorch
- Embedding Framework: Sentence Transformers, Colpali Engine

### Running Port and Performance Requirements

Default port: `localhost: 9090`

GPU Requirements: NVIDIA GPU (⩾ 7GB recommended), CUDA 11.3+

Python Requirements: Python 3.10+

### Configuration

Configuration Settings are all in config.py

| Parameter | Default Value | Description |
|---|---|---|
| MODE | 0 | RAG mode: 0 = pure-text scenario; 1 = multimodal scenario |
| EMBEDDED_DB_PATH | ./kb_test.db | Path to the embedded Milvus database file |
| CHUNK_EMBED_DIM | 384 | Embedding dimension for text chunks |

| | | |
|---|---|---|
| PAGE_EMBED_DIM | 128 | Embedding dimension for page-level (image) embeddings |
| IMG_DIR | ./imgs | Directory to store images generated from PDF files |

### How to start

Set all configuration settings in config.py

Install all dependencies and start service

```
$ cd rag
$ conda create -n rag python=3.12
$ conda activate rag
$ conda install -c conda-forge poppler

$ pip install -r requirements.txt
$ python app.py
```

## 6. Installation of TTS Models

### Running Port and Performance Requirements

Default port: `localhost: 5033`

Due to VRAM limitations, only **one** TTS model can run at a time in the current environment. And the VRAM usage for different TTS models is as follows:

| Model | RTF | Feature | VRAM Usage |
|---|---|---|---|
| EdgeTTS | 0.2 | Timbre switch | 0 GB (Web API) |
| Tacotron2 | 0.08–0.12 | Timbre switch | 1 GB |

|  |  |  |  |
|---|---|---|---|
| GPT-SoViTs | 0.4–0.6 | Zero-shot clone | 2.2 GB |
| CosyVoice2-0.5B | 0.6–0.8 | Zero-shot clone | 3.5 GB |

## File Structure

The storage structure of TTS models is as follows:

```
MODEL_NAME/
├──── env.sh  # List  of environment setup commands
├────logs/  # running logs storage
|    ├────...
├────output/  # temp output storage
|    ├────...
└────MODEL_FILE/
     ├────pretrained_models/   # Model  parameters and related files
     ├────server.py    # TTS model  audio generation server
     └────other related files…
```

Each model directory contains:

- Code repository required for running the model
- Training parameters
- Service script server.py

Location differences:

- **Tacotron** and **EdgeTTS**: server.py is located inside the model' s directory
- **CosyVoice** and **SoViTs**: server.py must be placed in the root directory of the code repository

These repositories are usually downloaded from GitHub.

The download commands can be found in the `env.sh` file inside each model' s folder.

### Dependencies and Installation

1. Find Dependency Installation Commands

- Navigate to the model' s folder and locate the `env.sh` file
- File `env.sh` contains commands for creating a virtual environment and installing basic dependencies
- Execute the commands one by one in the terminal
- Do not run `env.sh` directly

2. Install Full Dependencies

- Activate the virtual environment for the model
- Go to the requirements folder
- Find the subfolder named after the model
- Run the installation command to install all required dependencies

```
$ cd tts/requirments/MODEL_NAME
$ conda env create -f environment.yml
```

3. Quick Installation Script

Due to the framework design, the TTS server is managed entirely by the Talking Head Management module for starting, editing, and stopping services

This document does not provide a direct startup method.

For detailed instructions, refer to:

- API documentation
- Section 2.8 – Installation of Talking Head Management

## 7. Installation of Lip-sync and RTC

### Installation Requirements

GPU Requirements:

NVIDIA GPU, CUDA 11.3 or higher, at least 24GB of video memory

Network Requirements:

Stable internet connection with an uplink bandwidth of at least 5Mbps

Follows are the Installation Commands

## Configure the LiveTalking module

All lip-sync related codes are in the lip-sync folder. The following operations are run in the lip-sync folder by default.

```
# create conda env
conda create -n nerfstream python=3.10
conda activate nerfstream

# install pytorch,
# choose the install cmd according to  your cuda version by
# <https://pytorch.org/get-started/previous-versions/>
conda install pytorch==2.5.0  torchvision==0.20.0 torchaudio==2.5.0
pytorch-cuda=12.4 -c pytorch -c nvidia

# Install moudle for MuseTalk
conda install ffmpeg
pip install --no-cache-dir -U  openmim
mim install mmengine
mim install "mmcv==2.1.0"
# Compilation here will take up a lot  of CPU,
# pay attention to monitor CPU usage
mim install  "mmdet==3.2.0"
mim install  "mmpose>=1.1.0"

# download model file according to  README in lip-sync
```

After the installation is complete, modify the lip-sync.json file, change conda_init to the conda activation script, conda_env to the nerfstream environment location, and working_directory to the directory where the lip-sync module is located

## Configure the video background blur module

This module is used to implement the background blur function when the user uploads a video file. This module uses the grpc service call, and the grpc server needs to be started. This module code is in the blur folder of lip-sync

Firstly, download human_segmentation_pphumanseg_2023mar.onnx

from GitHub , then run these commands on the terminal

```
$ conda activate nerfstream
$ cd burr
```

## Configure the MuseTalk module

This project uses the MuseTalk project to generate inference intermediate files. For specific configuration methods, please refer to the MuseTalk project

https://github.com/TMElyralab/MuseTalk

```
# get musetalk
git clone  https://github.com/TMElyralab/MuseTalk.git
cd MuseTalk

# create a conda env according to  guide in musetalk
conda create -n MuseTalk python==3.10
conda activate MuseTalk
```

```

# Option 1: Using pip
pip install torch==2.0.1  torchvision==0.15.2 torchaudio==2.0.2 --index-url
https://download.pytorch.org/whl/cu118

# Option 2: Using conda
conda install pytorch==2.0.1  torchvision==0.15.2 torchaudio==2.0.2
pytorch-cuda=11.8 -c pytorch -c nvidia

# install required packages
pip install -r requirements.txt

pip install --no-cache-dir -U openmim
mim install mmengine
mim install "mmcv==2.0.1"
mim install "mmdet==3.1.0"
mim install "mmpose==1.1.0"

# set FFMPEG
export FFMPEG_PATH=/path/to/ffmpeg
# Example:
export  FFMPEG_PATH=/musetalk/ffmpeg-4.4-amd64-static

# download weights
bash ./download_weights.sh
```

After installing MuseTalk, change muse_conda_env in lip-sync.json to the path of the conda environment in MuseTalk, ffmpeg_path to the path of the installed ffmpeg, and muse_talk_base to the path of the folder where MuseTalk is located.

## How to start

For the lip-sync module, you need to start the GRPC server that provides the blurred background service and the server that provides the lip-sync backend service. The ports occupied by the two servers can be modified in lip-sync.json.

The default ports are lip-sync backend service: 8205, blurred background service: 23002

```
# Start burr_server, if port 23002 is  bind, use other port
$ cd burr
$ conda activate nerfstream
$ python burr_server.py

# start lip-sync backend server
$ cd ../
$ python live_server.py
```

## 8. Installation of Talking Head management

### Running Port and Performance Requirements

Default port: `localhost: 8124`

According to client requirements, the voice and tutors’ faces are bound to avatars (talking head). Students are only allowed to select an avatar, which means that students cannot directly specify the TTS model, voice, or tutor’s face.

Due to GPU memory limitations, only one avatar can be active at a time. If a user needs to switch avatars, they must wait for the current avatar to shut down before starting the next one. This process takes approximately **one minute**.

### Dependencies and Installation

The service depends on three JSON files located in the *tts/* directory:

**model_info.json** – Stores the list of supported TTS models and their metadata.

The basic structure of this config file is:

```
{
   "tacotron": {
      "full_name": "TacoTron2",
      "clone": false,
      "license": "MIT",
      "env_path": "/path/to/model/env ",
      "server_path": "/path/to/tts/server.py",
      "status": "active",
      "timbres": [
       "Default",
       "MaleA",
       "MaleB",
       "FemaleA",
       "FemaleB"
      ],
      "cur_timbre": "Default"
   },
   ...
}
```

**config.json** – Stores the current runtime configuration of the active TTS service.

The basic structure of this config file is:

```
{
   "tts_server_port": 5033,
   "tts_server_use_gpu": true,
   "current_tts_server": "edgeTTS",
   "tts_server_pid": 3514099,
   "tts_timbre": "FemaleA"
}
```

**avatar_info.json** – Stores the list of supported avatars and their metadata.

The basic structure of this config file is:

```
{
    "test_avatar": {
        "description": "This is a test Avatar",
        "status": "active",
        "prompt_face": "path/to/prompt/face",
        "avatar_preview": " path/to/prompt/preview ",
        "avatar_blur": true,
        "avatar_model": "MuseTalk",
        "clone": false,
        "tts_model": "edgeTTS",
        "timbre": "Default",
        "prompt_voice": " path/to/prompt/voice "
    },
    ...
}
```

If these files are missing, the server will create them automatically. However, the generated environment settings, model paths, and configurations may be incorrect and must be modified manually.

## Environment Setup

The server’ s dependencies can be found in the **requirements/base** folder.

Run the following command to install all dependencies for the model:

```
$ cd tts/requirments/base
$ conda env create -f environment.yml
```

After installation, activate the specified virtual environment.

## How to start

Once the environment dependencies are installed, switch to the designated virtual environment and run the model management server.

The Model Management server is stored in the *tts/* directory and is named *tts.py*. This script is an improved version of the TTS Model Management service.

```
$ cd tts/
$ uvicorn tts:app --host 0.0.0.0 --port 8204
```