# **AI FusionMind MultiModel**

## 

## **Tamil to English translation Model:**

### Model:

[Model --- Link](https://huggingface.co/facebook/mbart-large-50-many-to-one-mmt)

### Transformer Module:

* ***from transformers import MBartForConditionalGeneration, MBart50TokenizerFast***

This uses the Hugging face Transformers Library for that is imported the two Classes **MBartForConditionalGeneration, MBart50TokenizerFast**

1. **MBartForConditionalGeneration:** This class is used for generating sequences based on input data. It is specifically designed for tasks where the output is conditioned on the input, such as translation or summarization. The model uses a language modeling head, which allows it to generate text based on the input it receives.
2. **MBart50TokenizerFast:** This tokenizer is responsible for converting text into a format that the mBART model can understand. It tokenizing the input text, handles special tokens (such as beginning-of-sequence and end-of-sequence tokens), and prepares the input for the model. The "Fast" version of the tokenizer is optimized for speed and performance.

### Translation code for Tamil to English translation:

import requests

import json

# Set the Hugging Face API URL and token

API\_URL = "https://api-inference.huggingface.co/models/facebook/mbart-large-50-many-to-one-mmt"

HUGGINGFACE\_TOKEN = "Token"

headers = {"Authorization": f"Bearer {HUGGINGFACE\_TOKEN}"}

text = "செயற்கை நுண்ணறிவு"

payload = {

"inputs": text

}

# Make the request to the API

response = requests.post(API\_URL, headers=headers, json=payload)

# Check if the request was successful

if response.status\_code == 200:

# Parse the response

result = response.json()

translated\_text = result[0]['generated\_text']

print("Translated Text:", translated\_text)

else:

print(f"Error {response.status\_code}: {response.text}")

First importing the requests and json module.

Requests Module: It is used for establishing the connections between the client(web browser) and the server (the website or thing that we are trying to access)

Json Module: It is used to parse the response

#### Accessing the Model using the Hugging face Access token:

API\_URL = "https://api-inference.huggingface.co/models/facebook/mbart-large-50-many-to-one-mmt"

HUGGINGFACE\_TOKEN = "Token"

headers = {"Authorization": f"Bearer {HUGGINGFACE\_TOKEN}"}

text = "செயற்கை நுண்ணறிவு"

In api\_url I have given the url of the model I need to access and in the Huggingface\_token variable I have given the Access Token key and in headers fed that tokens for the Authorization purpose to access the website and finally given the text needed to translate.

#### Making the Request to the API:

payload = {

"inputs": text

}

# Make the request to the API

response = requests.post(API\_URL, headers=headers, json=payload)

Inside the payload the input text will be fed and inside the response variable the we are posting the request to the server with url , access key and the input text and the output will be captured in the response variable.

#### Status code

if response.status\_code == 200:

# Parse the response

result = response.json()

translated\_text = result[0]['generated\_text']

print("Translated Text:", translated\_text)

else:

print(f"Error {response.status\_code}: {response.text}")

In this case the status code is 200 i.e Success means the response will be parsed in the json format and the translated output will be extracted and stored and printed.

If the response fails, it will print the error response status code and the text of that error.

## Content Generation Model:

[Groq Playground](https://console.groq.com/playground)

import requests

# Setting Groq API key directly in the code

api\_key = "Token"

# API endpoint

url = "https://api.groq.com/openai/v1/chat/completions"

# Headers

headers = {

"Authorization": f"Bearer {api\_key}",

"Content-Type": "application/json"

}

# Request payload for educational and creative content generation

payload = {

#"model": "llama3-8b-8192"

#"model": "llama-3.1-70b-versatile", # The model

#"model": "gemma2-9b-it",

"model": "mixtral-8x7b-32768",# The model

"messages": [

{"role": "system", "content": "You are a creative and insightful writer who excels at generating engaging and original educational content. Your style makes learning fun, imaginative, and easy to understand."},

{"role": "user", "content": "Write a content about Artificical Intelligence with in 300 tokens"}

],

"max\_tokens": 300 # Enough space for educational explanations

}

# Send POST request to Groq API

response = requests.post(url, json=payload, headers=headers)

# Check if the request was successful

if response.status\_code == 200:

# Parse the JSON response

result = response.json()

# Extract and print the generated text

generated\_text = result['choices'][0]['message']['content']

print("Generated text:", generated\_text)

else:

print("Error:", response.status\_code, response.text)

import requests

# Setting Groq API key directly in the code

api\_key = "Token"

# API endpoint

url = "https://api.groq.com/openai/v1/chat/completions"

# Headers

headers = {

"Authorization": f"Bearer {api\_key}",

"Content-Type": "application/json"

}

In this I have used the 4 different models for the Text generation and I have picked the best and well suited model for content generation.

* llama-3.1-70b-versatile
* gemma-2-9b-it
* Llama3-8b-8192
* Mixtral-8x7b-32768

Out of these four models after continuous testing of the model I have decided **llama-3.1-70b-versatile** might be the best model for the content generation because this model generates the heading wise contents which is very much important for the content based results so I have decided to go with this model.

### API and Authorization of the for GROQ API Integration:

import requests

# Setting Groq API key directly in the code

api\_key = "Token"

# API endpoint

url = "https://api.groq.com/openai/v1/chat/completions"

# Headers

headers = {

"Authorization": f"Bearer {api\_key}",

"Content-Type": "application/json"

}

Firstly importing the request module for the posting the request to the groq and specifying the api for accessing the groq models and inside the headers there is the content-Type where I was specifying the how I am going to give the input in which format in this i have specified since it was a json.

### Inputting the model, messages, role and Tokens:

payload = {

#"model": "llama3-8b-8192"

#"model": "llama-3.1-70b-versatile", # The model

#"model": "gemma2-9b-it",

"model": "mixtral-8x7b-32768",# The model

"messages": [

{"role": "system", "content": "You are a creative and insightful writer who excels at generating engaging and original educational content. Your style makes learning fun, imaginative, and easy to understand."},

{"role": "user", "content": "Write a content about Artificial Intelligence with in 300 tokens"}

],

"max\_tokens": 300 # Enough space for educational explanations

}

In this I have tried with the 4 different model as i said before and in **message** key I have specified two roles one is the **system** and another is the **users** input since I need a output as a content based so I have specified how the model need to be in the system role and in **user role** I have given the prompt for generating the content that I need and finally I have mention a max **token** as 300 so the model won’t generate a content more than 300 words.

### Finally the Response status:

response = requests.post(url, json=payload, headers=headers)

# Check if the request was successful

if response.status\_code == 200:

# Parse the JSON response

result = response.json()

# Extract and print the generated text

generated\_text = result['choices'][0]['message']['content']

print("Generated text:", generated\_text)

else:

print("Error:", response.status\_code, response.text)

In this case the status code is 200 i.e Success means the response will be parsed in the json format and the generated output will be extracted and stored and printed.

If the response fails, it will print the error response status code and the text of that error.

## Image prompt generation Model:

[Groq Playground](https://console.groq.com/playground)

import requests

# Setting Groq API key directly in the code

api\_key = "Token"

# API endpoint

url = "https://api.groq.com/openai/v1/chat/completions"

# Headers

headers = {

"Authorization": f"Bearer {api\_key}",

"Content-Type": "application/json"

}

# Request payload for educational and creative content generation

payload = {

#"model": "llama3-8b-8192",

#"model": "llama-3.1-70b-versatile", # The model

#"model": "gemma2-9b-it",

"model": "mixtral-8x7b-32768",# The model

"messages": [

{"role": "system", "content": "You are a realistic,insightful,positive and professional Text to image prompt generator which generate a prompt for image generation"},

{"role": "user", "content": "create a text to image generation prompt about Artificial Intelligence with in 100 tokens"}

],

"max\_tokens": 30 # Enough space for educational explanations

}

# Send POST request to Groq API

response = requests.post(url, json=payload, headers=headers)

# Check if the request was successful

if response.status\_code == 200:

# Parse the JSON response

result = response.json()

# Extract and print the generated text

generated\_text = result['choices'][0]['message']['content']

print("Generated text:", generated\_text)

else:

print("Error:", response.status\_code, response.text)

### Code Explanation:

This is the same code that I have used for the content generation but for this I have used **mixtral-8x7b-32768** model for the Image prompt generations and I have used 30 tokens for generating the text prompt and I have changed the system role so it will behave like the professional image prompt generator.

## Text to Image Generation Model

[Hugging Face Models](https://huggingface.co/black-forest-labs/FLUX.1-schnell)

import requests

from PIL import Image

from io import BytesIO

# Set the API URL and Hugging Face token

#API\_URL = "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-schnell"

API\_URL = "https://api-inference.huggingface.co/models/CompVis/stable-diffusion-v1-4"

#API\_URL = "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-dev"

HUGGINGFACE\_TOKEN = "Token"

headers = {"Authorization": f"Bearer {HUGGINGFACE\_TOKEN}"}

def query(payload):

response = requests.post(API\_URL, headers=headers, json=payload)

if response.status\_code == 200:

return response.content

else:

print(f"Error {response.status\_code}: {response.text}")

return None

# Input text prompt for generating the image

text\_prompt = "Create an image of a futuristic robot interacting with humans in a modern city, showcasing advanced AI technology"

# Payload for the API request

data = {

"inputs": text\_prompt

}

# Query the API with the text prompt

image\_data = query(data)

# If the response contains image data, display it

if image\_data:

image = Image.open(BytesIO(image\_data))

image.show() # Opens the image in the default viewer

image.save("generated\_image.png") # Optionally save the image

### Importing the Modules:

import requests

from PIL import Image

from io import BytesIO

**requests:** Ideal for interacting with APIs, making HTTP requests, downloading files, or interacting with web services. In this case, it sends a text prompt to an image generation model via POST request.

**Pillow (PIL):** Used for working with images in Python. You can open, manipulate, and save images. It's useful for handling image data returned from APIs, applying filters, resizing images, or creating new ones.

**io.BytesIO:** Useful for working with in-memory binary streams, such as converting raw binary data (like an image returned by an API) into an object that can be opened as a file in Python.

### API Setting and Hugging Face Interaction:

#API\_URL = "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-schnell"

API\_URL = "https://api-inference.huggingface.co/models/CompVis/stable-diffusion-v1-4"

#API\_URL = "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-dev"

HUGGINGFACE\_TOKEN = "Token"

headers = {"Authorization": f"Bearer {HUGGINGFACE\_TOKEN}"}

For Image generation I have taken 3 different model for generating the image namely Flux.1-schnell, Stable-diffusion,Flux.1-dev are the model which I have used for the image generation task and since it was in a Hugging face I have used a Hugging face Access token

### Image Prompt and Execution:

def query(payload):

response = requests.post(API\_URL, headers=headers, json=payload)

if response.status\_code == 200:

return response.content

else:

print(f"Error {response.status\_code}: {response.text}")

return None

# Input text prompt for generating the image

text\_prompt = "Create an image of a futuristic robot interacting with humans in a modern city, showcasing advanced AI technology"

# Payload for the API request

data = {

"inputs": text\_prompt

}

# Query the API with the text prompt

image\_data = query(data)

In here the **text\_prompt** is the variable where we want to give a image prompt to generate and once we given the input means it will fed into the data dictionary and the dictionary will be posted as a request as previously said if the response is success means the image data will be send or else the error status code with the error message will be sent.

### Getting and displaying the image Data:

# If the response contains image data, display it

if image\_data:

image = Image.open(BytesIO(image\_data))

image.show() # Opens the image in the default viewer

image.save("generated\_image.png") # Optionally save the image

If the image data is successful means the BytesIo will convert the Binary coded image data into an image object which can be opened and the image data will be opened and stored in the image variable and that image will be shown and saved for feature use.

## Integrating all the model together and testing:

import requests

import json

from PIL import Image

from io import BytesIO

# Hugging Face API Token

HUGGINGFACE\_TOKEN = "Token"

headers\_hf = {"Authorization": f"Bearer {HUGGINGFACE\_TOKEN}"}

# Groq API Token

GROQ\_API\_KEY = "Token"

headers\_groq = {

"Authorization": f"Bearer {GROQ\_API\_KEY}",

"Content-Type": "application/json"

}

# Translation Model API URL (Tamil to English)

translation\_url = "https://api-inference.huggingface.co/models/facebook/mbart-large-50-many-to-one-mmt"

# Text-to-Image Model API URL

image\_generation\_url = "https://api-inference.huggingface.co/models/CompVis/stable-diffusion-v1-4"

# Function to query Hugging Face translation model

def translate\_text(text):

payload = {"inputs": text}

response = requests.post(translation\_url, headers=headers\_hf, json=payload)

if response.status\_code == 200:

result = response.json()

translated\_text = result[0]['generated\_text']

return translated\_text

else:

print(f"Translation Error {response.status\_code}: {response.text}")

return None

# Function to query Groq content generation model

def generate\_content(english\_text):

url = "https://api.groq.com/openai/v1/chat/completions"

payload = {

"model": "mixtral-8x7b-32768",

"messages": [

{"role": "system", "content": "You are a creative and insightful writer."},

{"role": "user", "content": f"Write educational content about {english\_text} within 300 tokens."}

],

"max\_tokens": 300

}

response = requests.post(url, json=payload, headers=headers\_groq)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

print(f"Content Generation Error: {response.status\_code}")

return None

# Function to generate image prompt

def generate\_image\_prompt(english\_text):

payload = {

"model": "mixtral-8x7b-32768",

"messages": [

{"role": "system", "content": "You are a professional Text to image prompt generator."},

{"role": "user", "content": f"Create a text to image generation prompt about {english\_text} within 30 tokens."}

],

"max\_tokens": 30

}

response = requests.post("https://api.groq.com/openai/v1/chat/completions", json=payload, headers=headers\_groq)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

print(f"Prompt Generation Error: {response.status\_code}")

return None

# Function to generate an image from the prompt

def generate\_image(image\_prompt):

data = {"inputs": image\_prompt}

response = requests.post(image\_generation\_url, headers=headers\_hf, json=data)

if response.status\_code == 200:

return response.content

else:

print(f"Image Generation Error {response.status\_code}: {response.text}")

return None

# Main function to handle multimodal pipeline

def multimodal\_pipeline():

# Step 1: Get Tamil input

tamil\_input = input("Enter Tamil text (or press Enter to input in English): ").strip()

if tamil\_input:

# Step 2: Attempt translation

english\_text = translate\_text(tamil\_input)

if english\_text:

print("Translated Text:", english\_text)

else:

# Translation failed, ask for English input

english\_text = input("Translation failed. Please enter English input: ").strip()

else:

# No Tamil input provided, ask for English input

english\_text = input("Please enter English input: ").strip()

# Step 3: Generate educational content

content\_output = generate\_content(english\_text)

print("Content Generation Output:", content\_output)

# Step 4: Generate image prompt and image

image\_prompt = generate\_image\_prompt(english\_text)

image\_data = generate\_image(image\_prompt)

if image\_data:

# Step 5: Display and save the image

image = Image.open(BytesIO(image\_data))

image.show() # Opens the image in the default viewer

image.save("generated\_image.png") # Optionally save the image

# Run the multimodal pipeline

multimodal\_pipeline()

### Importing all the modules and loading the model though API inference URL:

import requests

import json

from PIL import Image

from io import BytesIO

# Hugging Face API Token

HUGGINGFACE\_TOKEN = "Token"

headers\_hf = {"Authorization": f"Bearer {HUGGINGFACE\_TOKEN}"}

# Groq API Token

GROQ\_API\_KEY = "Token"

headers\_groq = {

"Authorization": f"Bearer {GROQ\_API\_KEY}",

"Content-Type": "application/json"

}

# Translation Model API URL (Tamil to English)

translation\_url = "https://api-inference.huggingface.co/models/facebook/mbart-large-50-many-to-one-mmt"

# Text-to-Image Model API URL

image\_generation\_url = "https://api-inference.huggingface.co/models/CompVis/stable-diffusion-v1-4"

As I said previously I imported the requests for connection between the client and the server and json for parsing the input in the json format and PIL for the open,manipulating and saving the image file and finally the BytesIO for converting the Binary data into an image format.

### Translation Function:

def translate\_text(text):

payload = {"inputs": text}

response = requests.post(translation\_url, headers=headers\_hf, json=payload)

if response.status\_code == 200:

result = response.json()

translated\_text = result[0]['generated\_text']

return translated\_text

else:

print(f"Translation Error {response.status\_code}: {response.text}")

return None

In this the tamil input will be fed as an argument and the request will be sent if the response is successful means it will return the translated text or else it will print the error with the status code and return none.

### Content generation function:

def generate\_content(english\_text):

url = "https://api.groq.com/openai/v1/chat/completions"

payload = {

"model": "mixtral-8x7b-32768",

"messages": [

{"role": "system", "content": "You are a creative and insightful writer."},

{"role": "user", "content": f"Write educational content about {english\_text} within 300 tokens."}

],

"max\_tokens": 300

}

response = requests.post(url, json=payload, headers=headers\_groq)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

print(f"Content Generation Error: {response.status\_code}")

return None

In this the api url will be given and the role of the system and also for the user the role has been mentioned after that the max token is set as 300 and finally if the content is generated successfully means the result will be fed or else the error status code will be displayed and none type is returned.

### Image Prompt generation Function:

def generate\_image\_prompt(english\_text):

payload = {

"model": "mixtral-8x7b-32768",

"messages": [

{"role": "system", "content": "You are a professional Text to image prompt generator."},

{"role": "user", "content": f"Create a text to image generation prompt about {english\_text} within 30 tokens."}

],

"max\_tokens": 30

}

response = requests.post("https://api.groq.com/openai/v1/chat/completions", json=payload, headers=headers\_groq)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

print(f"Prompt Generation Error: {response.status\_code}")

return None

In this the prompt for the image will be generated here also the same process is followed the role for the system and user is mentioned and max token is said to be 30 and the result will be displayed if it was successful or else it will return the none type.

### Image Generation Model function:

def generate\_image(image\_prompt):

data = {"inputs": image\_prompt}

response = requests.post(image\_generation\_url, headers=headers\_hf, json=data)

if response.status\_code == 200:

return response.content

else:

print(f"Image Generation Error {response.status\_code}: {response.text}")

return None

In this previously generated image prompt will be given as an input and that input will be posted through the request and if the model sends the binary code it will return it or else it will print the error status code and return none.

### Function that integrate all the model’s and give the Output:

def multimodal\_pipeline():

# Step 1: Get Tamil input

tamil\_input = input("Enter Tamil text (or press Enter to input in English): ").strip()

if tamil\_input:

# Step 2: Attempt translation

english\_text = translate\_text(tamil\_input)

if english\_text:

print("Translated Text:", english\_text)

else:

# Translation failed, ask for English input

english\_text = input("Translation failed. Please enter English input: ").strip()

else:

# No Tamil input provided, ask for English input

english\_text = input("Please enter English input: ").strip()

# Step 3: Generate educational content

content\_output = generate\_content(english\_text)

print("Content Generation Output:", content\_output)

# Step 4: Generate image prompt and image

image\_prompt = generate\_image\_prompt(english\_text)

image\_data = generate\_image(image\_prompt)

if image\_data:

# Step 5: Display and save the image

image = Image.open(BytesIO(image\_data))

image.show() # Opens the image in the default viewer

image.save("generated\_image.png") # Optionally save the image

# Run the multimodal pipeline

multimodal\_pipeline()

Firstly it will ask to enter the tamil input if the tamil text is entered means it will calls the **translation function** and the translated output will be given that english text will be fed as a argument to the **generate\_content function** and the generated output will be printed again the english text will fed into the **generate image prompt function** that will create a short and concise prompt for the image generation that will used to create a image and the **generated image** will be displayed and saved for future use while calling the multimodel\_pipeline all this function will get executed.

## Streamlite Integrated model —> Version 1 with API integration:

import streamlit as st

import requests

import os

# Fetch Hugging Face and Groq API keys from secrets

Transalate\_token = os.getenv('Translate')

Image\_Token = os.getenv('Image\_generation')

Content\_Token = os.getenv('ContentGeneration')

Image\_prompt\_token = os.getenv('Prompt\_generation')

# API Headers

Translate = {"Authorization": f"Bearer {Transalate\_token}"}

Image\_generation = {"Authorization": f"Bearer {Image\_Token}"}

Content\_generation = {

"Authorization": f"Bearer {Content\_Token}",

"Content-Type": "application/json"

}

Image\_Prompt = {

"Authorization": f"Bearer {Image\_prompt\_token}",

"Content-Type": "application/json"

}

# Translation Model API URL (Tamil to English)

translation\_url = "https://api-inference.huggingface.co/models/facebook/mbart-large-50-many-to-one-mmt"

# Text-to-Image Model API URL

image\_generation\_url = "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-schnell"

# Function to query Hugging Face translation model

def translate\_text(text):

payload = {"inputs": text}

response = requests.post(translation\_url, headers=Translate, json=payload)

if response.status\_code == 200:

result = response.json()

translated\_text = result[0]['generated\_text']

return translated\_text

else:

st.error(f"Translation Error {response.status\_code}: {response.text}")

st.write(f'Please try after sometime 😥😥😥')

return None

# Function to query Groq content generation model

def generate\_content(english\_text, max\_tokens, temperature):

url = "https://api.groq.com/openai/v1/chat/completions"

payload = {

"model": "llama-3.1-70b-versatile",

"messages": [

{"role": "system", "content": "You are a creative and insightful writer."},

{"role": "user", "content": f"Write educational content about {english\_text} within {max\_tokens} tokens."}

],

"max\_tokens": max\_tokens,

"temperature": temperature

}

response = requests.post(url, json=payload, headers=Content\_generation)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

st.error(f"Content Generation Error: {response.status\_code}")

return None

# Function to generate image prompt

def generate\_image\_prompt(english\_text):

payload = {

"model": "mixtral-8x7b-32768",

"messages": [

{"role": "system", "content": "You are a professional Text to image prompt generator."},

{"role": "user", "content": f"Create a text to image generation prompt about {english\_text} within 30 tokens."}

],

"max\_tokens": 30

}

response = requests.post("https://api.groq.com/openai/v1/chat/completions", json=payload, headers=Image\_Prompt)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

st.error(f"Prompt Generation Error: {response.status\_code}")

return None

# Function to generate an image from the prompt

def generate\_image(image\_prompt):

data = {"inputs": image\_prompt}

response = requests.post(image\_generation\_url, headers=Image\_generation, json=data)

if response.status\_code == 200:

return response.content

else:

st.error(f"Image Generation Error {response.status\_code}: {response.text}")

return None

# Main Streamlit app

def main():

# Custom CSS for background, borders, and other styling

st.markdown(

"""

<style>

body {

background-image: url('https://wallpaperaccess.com/full/1567666.png');

background-size: cover;

}

.reportview-container {

background: rgba(255, 255, 255, 0.85);

padding: 2rem;

border-radius: 10px;

box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.1);

}

.result-container {

border: 2px solid #4CAF50;

padding: 20px;

border-radius: 10px;

margin-top: 20px;

animation: fadeIn 2s ease;

}

@keyframes fadeIn {

0% { opacity: 0; }

100% { opacity: 1; }

}

.stButton button {

background-color: #4CAF50;

color: white;

border-radius: 10px;

padding: 10px;

}

.stButton button:hover {

background-color: #45a049;

transform: scale(1.05);

transition: 0.2s ease-in-out;

}

</style>

""", unsafe\_allow\_html=True

)

st.title("🅰️ℹ️ FusionMind ➡️ Multimodal")

# Sidebar for temperature and token adjustment

st.sidebar.header("Settings")

temperature = st.sidebar.slider("Select Temperature", 0.1, 1.0, 0.7)

max\_tokens = st.sidebar.slider("Max Tokens for Content Generation", 100, 400, 200)

# Suggested inputs

st.write("## Suggested Inputs")

suggestions = ["தரவு அறிவியல்", "புதிய திறன்களைக் கற்றுக்கொள்வது எப்படி", "ராக்கெட் எப்படி வேலை செய்கிறது"]

selected\_suggestion = st.selectbox("Select a suggestion or enter your own:", [""] + suggestions)

# Input box for user

tamil\_input = st.text\_input("Enter Tamil text (or select a suggestion):", selected\_suggestion)

if st.button("Generate"):

# Step 1: Translation (Tamil to English)

if tamil\_input:

st.write("### Translated English Text:")

english\_text = translate\_text(tamil\_input)

if english\_text:

st.success(english\_text)

# Step 2: Generate Educational Content

st.write("### Generated Educational Content:")

with st.spinner('Generating content...'):

content\_output = generate\_content(english\_text, max\_tokens, temperature)

if content\_output:

st.success(content\_output)

# Step 3: Generate Image from the prompt

st.write("### Generated Image:")

with st.spinner('Generating image...'):

image\_prompt = generate\_image\_prompt(english\_text)

image\_data = generate\_image(image\_prompt)

if image\_data:

st.image(image\_data, caption="Generated Image")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Since the model integration part is already I have explained much clearly now I will explain only about the streamlite part of the code.

### Streamlite Part:

st.title("🅰️ℹ️ FusionMind ➡️ Multimodal")

# Sidebar for temperature and token adjustment

st.sidebar.header("Settings")

temperature = st.sidebar.slider("Select Temperature", 0.1, 1.0, 0.7)

max\_tokens = st.sidebar.slider("Max Tokens for Content Generation", 100, 400, 200)

# Suggested inputs

st.write("## Suggested Inputs")

suggestions = ["தரவு அறிவியல்", "புதிய திறன்களைக் கற்றுக்கொள்வது எப்படி", "ராக்கெட் எப்படி வேலை செய்கிறது"]

selected\_suggestion = st.selectbox("Select a suggestion or enter your own:", [""] + suggestions)

# Input box for user

tamil\_input = st.text\_input("Enter Tamil text (or select a suggestion):", selected\_suggestion)

if st.button("Generate"):

# Step 1: Translation (Tamil to English)

if tamil\_input:

st.write("### Translated English Text:")

english\_text = translate\_text(tamil\_input)

if english\_text:

st.success(english\_text)

# Step 2: Generate Educational Content

st.write("### Generated Educational Content:")

with st.spinner('Generating content...'):

content\_output = generate\_content(english\_text, max\_tokens, temperature)

if content\_output:

st.success(content\_output)

# Step 3: Generate Image from the prompt

st.write("### Generated Image:")

with st.spinner('Generating image...'):

image\_prompt = generate\_image\_prompt(english\_text)

image\_data = generate\_image(image\_prompt)

if image\_data:

st.image(image\_data, caption="Generated Image")

if \_\_name\_\_ == "\_\_main\_\_":

main()

In this code I have first declared a title part I have given a Title as AI Fusion Mind Multimodal Since I have integrating the multiple model together I have given this name and the next step is in sidebar I have given a setting to adjust the content generation style of the model if the **temperature increases** the content will be more **creative** if the **temperature** is **less** means the content is **more realistic** and straight forward and next setting is the Token by adjusting the **token** we can able to adjust the output **number of words**.

We can give an Input in two different ways. One is we can give it through selecting the suggested input so else we can give our own input after the input we can click the generate button to generate a content.

After the tamil input is given and clicked the generate button the text will feed into a function and if it was translated successfully means the english text will be used for generating the english content. And if the content is generated means the english text will give into the generate image prompt function there the image prompt for the image generation will be created and finally the image will be created the also this steps will happens when I called the main function

## Final Streamlite and Hugging face Integrated code:

import streamlit as st

import requests

import os

# Fetch Hugging Face and Groq API keys from secrets

Transalate\_token = os.getenv('Translate')

Image\_Token = os.getenv('Image\_generation')

Content\_Token = os.getenv('ContentGeneration')

Image\_prompt\_token = os.getenv('Prompt\_generation')

# API Headers

Translate = {"Authorization": f"Bearer {Transalate\_token}"}

Image\_generation = {"Authorization": f"Bearer {Image\_Token}"}

Content\_generation = {

"Authorization": f"Bearer {Content\_Token}",

"Content-Type": "application/json"

}

Image\_Prompt = {

"Authorization": f"Bearer {Image\_prompt\_token}",

"Content-Type": "application/json"

}

# Translation Model API URL (Tamil to English)

translation\_url = "https://api-inference.huggingface.co/models/facebook/mbart-large-50-many-to-one-mmt"

# Text-to-Image Model API URLs

image\_generation\_urls = {

"black-forest-labs/FLUX.1-schnell": "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-schnell",

"CompVis/stable-diffusion-v1-4": "https://api-inference.huggingface.co/models/CompVis/stable-diffusion-v1-4",

"black-forest-labs/FLUX.1-dev": "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-dev"

}

# Default image generation model

default\_image\_model = "black-forest-labs/FLUX.1-schnell"

# Content generation models

content\_models = {

"llama-3.1-70b-versatile": "llama-3.1-70b-versatile",

"llama3-8b-8192": "llama3-8b-8192",

"gemma2-9b-it": "gemma2-9b-it",

"mixtral-8x7b-32768": "mixtral-8x7b-32768"

}

# Default content generation model

default\_content\_model = "llama-3.1-70b-versatile"

# Function to query Hugging Face translation model

def translate\_text(text):

payload = {"inputs": text}

response = requests.post(translation\_url, headers=Translate, json=payload)

if response.status\_code == 200:

result = response.json()

translated\_text = result[0]['generated\_text']

return translated\_text

else:

st.write(f'There is a issue in Translation model,Please reload the page and try with some other inputs or please try again later😥😥😥')

st.write(f'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

st.error(f"Translation Error {response.status\_code}: {response.text}")

return None

# Function to query Groq content generation model

def generate\_content(english\_text, max\_tokens, temperature, model):

url = "https://api.groq.com/openai/v1/chat/completions"

payload = {

"model": model,

"messages": [

{"role": "system", "content": "You are a creative and insightful writer."},

{"role": "user", "content": f"Write educational content about {english\_text} within {max\_tokens} tokens."}

],

"max\_tokens": max\_tokens,

"temperature": temperature

}

response = requests.post(url, json=payload, headers=Content\_generation)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

st.error(f"Content Generation Error: {response.status\_code}")

return None

# Function to generate image prompt

def generate\_image\_prompt(english\_text):

payload = {

"model": "mixtral-8x7b-32768",

"messages": [

{"role": "system", "content": "You are a professional Text to image prompt generator."},

{"role": "user", "content": f"Create a text to image generation prompt about {english\_text} within 30 tokens."}

],

"max\_tokens": 30

}

response = requests.post("https://api.groq.com/openai/v1/chat/completions", json=payload, headers=Image\_Prompt)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

st.error(f"Prompt Generation Error: {response.status\_code}")

return None

# Function to generate an image from the prompt

def generate\_image(image\_prompt, model\_url):

data = {"inputs": image\_prompt}

response = requests.post(model\_url, headers=Image\_generation, json=data)

if response.status\_code == 200:

return response.content

else:

st.error(f"Image Generation Error {response.status\_code}: {response.text}")

return None

# User Guide Section

def show\_user\_guide():

st.title("FusionMind User Guide")

st.write("""

### Welcome to the FusionMind User Guide!

### How to use this app:

1. \*\*Input Tamil Text\*\*:

- You can either select one of the suggested Tamil phrases or input your own text. The app primarily focuses on Tamil inputs, but it supports a wide range of other languages as well (see the list below).

2. \*\*Generate Translations\*\*:

- Once you've input your text, the app will automatically translate it to English. The translation model is a \*\*many-to-one model\*\*, meaning it can take input from various languages and translate it into English.

3. \*\*Generate Educational Content\*\*:

- After translating the text into English, the app will generate \*\*educational content\*\* based on the translated input. You can adjust the creativity of the content generation using the temperature slider, and control the length of the output with the token limit setting.

4. \*\*Generate Images\*\*:

- In addition to generating content, the app can also generate an \*\*image\*\* related to the translated content. You don’t need to worry about creating complex image prompts—FusionMind includes an automatic \*\*image prompt generator\*\* that will convert your input into a well-defined image prompt, ensuring better image generation results.

---

### Features:

- \*\*Multilingual Translation\*\*:

- FusionMind supports a \*\*many-to-one translation model\*\*, so you can input text in a wide variety of languages, not just Tamil. Below are the supported languages:

- \*\*Arabic (ar\_AR)\*\*, \*\*Czech (cs\_CZ)\*\*, \*\*German (de\_DE)\*\*, \*\*English (en\_XX)\*\*, \*\*Spanish (es\_XX)\*\*, \*\*Estonian (et\_EE)\*\*, \*\*Finnish (fi\_FI)\*\*, \*\*French (fr\_XX)\*\*, \*\*Gujarati (gu\_IN)\*\*, \*\*Hindi (hi\_IN)\*\*, \*\*Italian (it\_IT)\*\*, \*\*Japanese (ja\_XX)\*\*, \*\*Kazakh (kk\_KZ)\*\*, \*\*Korean (ko\_KR)\*\*, \*\*Lithuanian (lt\_LT)\*\*, \*\*Latvian (lv\_LV)\*\*, \*\*Burmese (my\_MM)\*\*, \*\*Nepali (ne\_NP)\*\*, \*\*Dutch (nl\_XX)\*\*, \*\*Romanian (ro\_RO)\*\*, \*\*Russian (ru\_RU)\*\*, \*\*Sinhala (si\_LK)\*\*, \*\*Turkish (tr\_TR)\*\*, \*\*Vietnamese (vi\_VN)\*\*, \*\*Chinese (zh\_CN)\*\*, \*\*Afrikaans (af\_ZA)\*\*, \*\*Azerbaijani (az\_AZ)\*\*, \*\*Bengali (bn\_IN)\*\*, \*\*Persian (fa\_IR)\*\*, \*\*Hebrew (he\_IL)\*\*, \*\*Croatian (hr\_HR)\*\*, \*\*Indonesian (id\_ID)\*\*, \*\*Georgian (ka\_GE)\*\*, \*\*Khmer (km\_KH)\*\*, \*\*Macedonian (mk\_MK)\*\*, \*\*Malayalam (ml\_IN)\*\*, \*\*Mongolian (mn\_MN)\*\*, \*\*Marathi (mr\_IN)\*\*, \*\*Polish (pl\_PL)\*\*, \*\*Pashto (ps\_AF)\*\*, \*\*Portuguese (pt\_XX)\*\*, \*\*Swedish (sv\_SE)\*\*, \*\*Swahili (sw\_KE)\*\*, \*\*Tamil (ta\_IN)\*\*, \*\*Telugu (te\_IN)\*\*, \*\*Thai (th\_TH)\*\*, \*\*Tagalog (tl\_XX)\*\*, \*\*Ukrainian (uk\_UA)\*\*, \*\*Urdu (ur\_PK)\*\*, \*\*Xhosa (xh\_ZA)\*\*, \*\*Galician (gl\_ES)\*\*, \*\*Slovene (sl\_SI)\*\*.

- \*\*Temperature Adjustment\*\*:

- You can adjust the \*\*temperature\*\* of the content generation. A \*\*higher temperature\*\* makes the content more creative and varied, while a \*\*lower temperature\*\* generates more focused and deterministic responses.

- \*\*Token Limit\*\*:

- Set the \*\*maximum number of tokens\*\* for content generation. This allows you to control the length of the generated educational content.

- \*\*Automatic Retries\*\*:

- If a translation request fails due to any reason, the app is designed to \*\*automatically retry\*\*, ensuring a smooth experience.

- \*\*Auto-Generated Image Prompts\*\*:

- One of the unique features of FusionMind is the \*\*auto-generated image prompts\*\*. Even if you're not experienced in creating detailed prompts for image generation, the app will take care of this for you. It automatically converts the translated text or content into a well-defined prompt that produces more accurate and high-quality images.

---

Enjoy the multimodal experience with \*\*FusionMind\*\* and explore its powerful translation, content generation, and image generation features!

""")

# Main Streamlit app

def main():

# Sidebar Menu

st.sidebar.title("FusionMind Options")

page = st.sidebar.radio("Select a page:", ["Main App", "User Guide"])

if page == "User Guide":

show\_user\_guide()

return

# Custom CSS for background, borders, and other styling

st.markdown(

"""

<style>

body {

background-image: url('https://wallpapercave.com/wp/wp4008910.jpg');

background-size: cover;

}

.reportview-container {

background: rgba(255, 255, 255, 0.85);

padding: 2rem;

border-radius: 10px;

box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.1);

}

.result-container {

border: 2px solid #4CAF50;

padding: 20px;

border-radius: 10px;

margin-top: 20px;

animation: fadeIn 2s ease;

}

@keyframes fadeIn {

0% { opacity: 0; }

100% { opacity: 1; }

}

.stButton button {

background-color: #4CAF50;

color: white;

border-radius: 10px;

padding: 10px;

}

.stButton button:hover {

background-color: #45a049;

transform: scale(1.05);

transition: 0.2s ease-in-out;

}

</style>

""", unsafe\_allow\_html=True

)

st.title("🅰️ℹ️ FusionMind ➡️ Multimodal")

# Sidebar for temperature, token adjustment, and model selection

st.sidebar.header("Settings")

temperature = st.sidebar.slider("Select Temperature", 0.1, 1.0, 0.7)

max\_tokens = st.sidebar.slider("Max Tokens for Content Generation", 100, 400, 200)

# Content generation model selection

content\_model = st.sidebar.selectbox("Select Content Generation Model", list(content\_models.keys()), index=0)

# Image generation model selection

image\_model = st.sidebar.selectbox("Select Image Generation Model", list(image\_generation\_urls.keys()), index=0)

# Reminder about model availability

st.sidebar.warning("Note: Based on availability, some models might not work. Please try another model if an error occurs.By default the perfect model is selected try with it and then experiment with different models")

# Suggested inputs

st.write("## Suggested Inputs")

suggestions = ["தரவு அறிவியல்", "உளவியல்", "ராக்கெட் எப்படி வேலை செய்கிறது"]

selected\_suggestion = st.selectbox("Select a suggestion or enter your own:", [""] + suggestions)

# Input box for user

tamil\_input = st.text\_input("Enter Tamil text (or select a suggestion):", selected\_suggestion)

if st.button("Generate"):

# Step 1: Translation (Tamil to English)

if tamil\_input:

st.write("### Translated English Text:")

english\_text = translate\_text(tamil\_input)

if english\_text:

st.success(english\_text)

# Step 2: Generate Educational Content

st.write("### Generated Content:")

with st.spinner('Generating content...'):

content\_output = generate\_content(english\_text, max\_tokens, temperature, content\_models[content\_model])

if content\_output:

st.success(content\_output)

# Step 3: Generate Image from the prompt

st.write("### Generated Image:")

with st.spinner('Generating image...'):

image\_prompt = generate\_image\_prompt(english\_text)

image\_data = generate\_image(image\_prompt, image\_generation\_urls[image\_model])

if image\_data:

st.image(image\_data, caption="Generated Image")

if \_\_name\_\_ == "\_\_main\_\_":

main()

I this I have introduced few things **User\_Guide page** for guiding the user for how to use the app and additionally in the sidebar I have added a select box where User can select the **multiple model** for the **content** and **image generation** and In getting the API key it need to be in secret so that it won’t be accessible by the normal users and for accessing the value I have used the **getenv function** what this function will do is once we have given he key of the secret that we have created to store the **Api key** It will take the value(API Key) and make it used for further process.

## Fully working App Video:

[FusionMind-Demo.mp4](https://drive.google.com/file/d/1JSqgtaBEujTvVvHHrDhDW9MqXWg9mhNQ/view?usp=drive_link)

## Streamlite Integrated model —> Version 2 with translated model integration Download:

import streamlit as st

import requests

import os

from transformers import MBartForConditionalGeneration, MBart50TokenizerFast

# API keys for other features (optional)

Image\_Token = os.getenv('Image\_generation')

Content\_Token = os.getenv('ContentGeneration')

Image\_prompt\_token = os.getenv('Prompt\_generation')

# API Headers for external services (optional)

Image\_generation = {"Authorization": f"Bearer {Image\_Token}"}

Content\_generation = {

"Authorization": f"Bearer {Content\_Token}",

"Content-Type": "application/json"

}

Image\_Prompt = {

"Authorization": f"Bearer {Image\_prompt\_token}",

"Content-Type": "application/json"

}

# Text-to-Image Model API URLs

image\_generation\_urls = {

"black-forest-labs/FLUX.1-schnell": "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-schnell",

"CompVis/stable-diffusion-v1-4": "https://api-inference.huggingface.co/models/CompVis/stable-diffusion-v1-4",

"black-forest-labs/FLUX.1-dev": "https://api-inference.huggingface.co/models/black-forest-labs/FLUX.1-dev"

}

# Default content generation model

content\_models = {

"llama-3.1-70b-versatile": "llama-3.1-70b-versatile",

"llama3-8b-8192": "llama3-8b-8192",

"gemma2-9b-it": "gemma2-9b-it",

"mixtral-8x7b-32768": "mixtral-8x7b-32768"

}

# Load the translation model and tokenizer locally

@st.cache\_resource

def load\_translation\_model():

model = MBartForConditionalGeneration.from\_pretrained("facebook/mbart-large-50-many-to-one-mmt")

tokenizer = MBart50TokenizerFast.from\_pretrained("facebook/mbart-large-50-many-to-one-mmt")

return model, tokenizer

# Function to perform translation locally

def translate\_text\_local(text):

model, tokenizer = load\_translation\_model()

inputs = tokenizer(text, return\_tensors="pt", max\_length=512, truncation=True)

translated\_tokens = model.generate(\*\*inputs, forced\_bos\_token\_id=tokenizer.lang\_code\_to\_id["en\_XX"])

translated\_text = tokenizer.batch\_decode(translated\_tokens, skip\_special\_tokens=True)[0]

return translated\_text

# Function to query Groq content generation model (optional)

def generate\_content(english\_text, max\_tokens, temperature, model):

url = "https://api.groq.com/openai/v1/chat/completions"

payload = {

"model": model,

"messages": [

{"role": "system", "content": "You are a creative and insightful writer."},

{"role": "user", "content": f"Write educational content about {english\_text} within {max\_tokens} tokens."}

],

"max\_tokens": max\_tokens,

"temperature": temperature

}

response = requests.post(url, json=payload, headers=Content\_generation)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

st.error(f"Content Generation Error: {response.status\_code}")

return None

# Function to generate image prompt (optional)

def generate\_image\_prompt(english\_text):

payload = {

"model": "mixtral-8x7b-32768",

"messages": [

{"role": "system", "content": "You are a professional Text to image prompt generator."},

{"role": "user", "content": f"Create a text to image generation prompt about {english\_text} within 30 tokens."}

],

"max\_tokens": 30

}

response = requests.post("https://api.groq.com/openai/v1/chat/completions", json=payload, headers=Image\_Prompt)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['message']['content']

else:

st.error(f"Prompt Generation Error: {response.status\_code}")

return None

# Function to generate an image from the prompt (optional)

def generate\_image(image\_prompt, model\_url):

data = {"inputs": image\_prompt}

response = requests.post(model\_url, headers=Image\_generation, json=data)

if response.status\_code == 200:

return response.content

else:

st.error(f"Image Generation Error {response.status\_code}: {response.text}")

return None

# User Guide Section

def show\_user\_guide():

st.title("FusionMind User Guide")

st.write("""

### Welcome to the FusionMind User Guide!

### How to use this app:

1. \*\*Input Tamil Text\*\*:

- You can either select one of the suggested Tamil phrases or input your own text. The app primarily focuses on Tamil inputs, but it supports a wide range of other languages as well (see the list below).

2. \*\*Generate Translations\*\*:

- Once you've input your text, the app will automatically translate it to English. The translation model is a \*\*many-to-one model\*\*, meaning it can take input from various languages and translate it into English.

3. \*\*Generate Educational Content\*\*:

- After translating the text into English, the app will generate \*\*educational content\*\* based on the translated input. You can adjust the creativity of the content generation using the temperature slider, and control the length of the output with the token limit setting.

4. \*\*Generate Images\*\*:

- In addition to generating content, the app can also generate an \*\*image\*\* related to the translated content. You don’t need to worry about creating complex image prompts—FusionMind includes an automatic \*\*image prompt generator\*\* that will convert your input into a well-defined image prompt, ensuring better image generation results.

---

### Features:

- \*\*Multilingual Translation\*\*:

- FusionMind supports a \*\*many-to-one translation model\*\*, so you can input text in a wide variety of languages, not just Tamil. Below are the supported languages:

- \*\*Arabic (ar\_AR)\*\*, \*\*Czech (cs\_CZ)\*\*, \*\*German (de\_DE)\*\*, \*\*English (en\_XX)\*\*, \*\*Spanish (es\_XX)\*\*, \*\*Estonian (et\_EE)\*\*, \*\*Finnish (fi\_FI)\*\*, \*\*French (fr\_XX)\*\*, \*\*Gujarati (gu\_IN)\*\*, \*\*Hindi (hi\_IN)\*\*, \*\*Italian (it\_IT)\*\*, \*\*Japanese (ja\_XX)\*\*, \*\*Kazakh (kk\_KZ)\*\*, \*\*Korean (ko\_KR)\*\*, \*\*Lithuanian (lt\_LT)\*\*, \*\*Latvian (lv\_LV)\*\*, \*\*Burmese (my\_MM)\*\*, \*\*Nepali (ne\_NP)\*\*, \*\*Dutch (nl\_XX)\*\*, \*\*Romanian (ro\_RO)\*\*, \*\*Russian (ru\_RU)\*\*, \*\*Sinhala (si\_LK)\*\*, \*\*Turkish (tr\_TR)\*\*, \*\*Vietnamese (vi\_VN)\*\*, \*\*Chinese (zh\_CN)\*\*, \*\*Afrikaans (af\_ZA)\*\*, \*\*Azerbaijani (az\_AZ)\*\*, \*\*Bengali (bn\_IN)\*\*, \*\*Persian (fa\_IR)\*\*, \*\*Hebrew (he\_IL)\*\*, \*\*Croatian (hr\_HR)\*\*, \*\*Indonesian (id\_ID)\*\*, \*\*Georgian (ka\_GE)\*\*, \*\*Khmer (km\_KH)\*\*, \*\*Macedonian (mk\_MK)\*\*, \*\*Malayalam (ml\_IN)\*\*, \*\*Mongolian (mn\_MN)\*\*, \*\*Marathi (mr\_IN)\*\*, \*\*Polish (pl\_PL)\*\*, \*\*Pashto (ps\_AF)\*\*, \*\*Portuguese (pt\_XX)\*\*, \*\*Swedish (sv\_SE)\*\*, \*\*Swahili (sw\_KE)\*\*, \*\*Tamil (ta\_IN)\*\*, \*\*Telugu (te\_IN)\*\*, \*\*Thai (th\_TH)\*\*, \*\*Tagalog (tl\_XX)\*\*, \*\*Ukrainian (uk\_UA)\*\*, \*\*Urdu (ur\_PK)\*\*, \*\*Xhosa (xh\_ZA)\*\*, \*\*Galician (gl\_ES)\*\*, \*\*Slovene (sl\_SI)\*\*.

- \*\*Temperature Adjustment\*\*:

- You can adjust the \*\*temperature\*\* of the content generation. A \*\*higher temperature\*\* makes the content more creative and varied, while a \*\*lower temperature\*\* generates more focused and deterministic responses.

- \*\*Token Limit\*\*:

- Set the \*\*maximum number of tokens\*\* for content generation. This allows you to control the length of the generated educational content.

- \*\*Auto-Generated Image Prompts\*\*:

- One of the unique features of FusionMind is the \*\*auto-generated image prompts\*\*. Even if you're not experienced in creating detailed prompts for image generation, the app will take care of this for you. It automatically converts the translated text or content into a well-defined prompt that produces more accurate and high-quality images.

---

Enjoy the multimodal experience with \*\*FusionMind\*\* and explore its powerful translation, content generation, and image generation features!

""")

# Main Streamlit app

def main():

# Sidebar Menu

st.sidebar.title("FusionMind Options")

page = st.sidebar.radio("Select a page:", ["Main App", "User Guide"])

if page == "User Guide":

show\_user\_guide()

return

st.title("🅰️ℹ️ FusionMind ➡️ Multimodal")

# Sidebar for temperature, token adjustment, and model selection

st.sidebar.header("Settings")

temperature = st.sidebar.slider("Select Temperature", 0.1, 1.0, 0.7)

max\_tokens = st.sidebar.slider("Max Tokens for Content Generation", 100, 400, 200)

# Content generation model selection

content\_model = st.sidebar.selectbox("Select Content Generation Model", list(content\_models.keys()), index=0)

# Image generation model selection

image\_model = st.sidebar.selectbox("Select Image Generation Model", list(image\_generation\_urls.keys()), index=0)

# Suggested inputs

st.write("## Suggested Inputs")

suggestions = ["தரவு அறிவியல்", "உளவியல்", "ராக்கெட் எப்படி வேலை செய்கிறது"]

selected\_suggestion = st.selectbox("Select a suggestion or enter your own:", [""] + suggestions)

# Input box for user

tamil\_input = st.text\_input("Enter Tamil text (or select a suggestion):", selected\_suggestion)

if st.button("Generate"):

# Step 1: Translation (Tamil to English)

if tamil\_input:

st.write("### Translated English Text:")

english\_text = translate\_text\_local(tamil\_input)

if english\_text:

st.success(english\_text)

# Step 2: Generate Educational Content

st.write("### Generated Content:")

with st.spinner('Generating content...'):

content\_output = generate\_content(english\_text, max\_tokens, temperature, content\_models[content\_model])

if content\_output:

st.success(content\_output)

# Step 3: Generate Image from the prompt (optional)

st.write("### Generated Image:")

with st.spinner('Generating image...'):

image\_prompt = generate\_image\_prompt(english\_text)

image\_data = generate\_image(image\_prompt, image\_generation\_urls[image\_model])

if image\_data:

st.image(image\_data, caption="Generated Image")

if \_\_name\_\_ == "\_\_main\_\_":

main()

### Why I developed the version 2:

Even Though the output generated in the first version is perfect and good there is one fault in it since I am using the API to integrating the Translation model there is a frequent error in the translation (model is loading - 92:3245) it may be due to that they are updating the model frequently which will affect my application so want I decide is the model size is around 2.5gb so I will download the model locally and process the next set so that this issue will be rectified.I have faced this issue only in Translation model other two model is working fine.

### Updated Code:

@st.cache\_resource

def load\_translation\_model():

model = MBartForConditionalGeneration.from\_pretrained("facebook/mbart-large-50-many-to-one-mmt")

tokenizer = MBart50TokenizerFast.from\_pretrained("facebook/mbart-large-50-many-to-one-mmt")

return model, tokenizer

# Function to perform translation locally

def translate\_text\_local(text):

model, tokenizer = load\_translation\_model()

inputs = tokenizer(text, return\_tensors="pt", max\_length=512, truncation=True)

translated\_tokens = model.generate(\*\*inputs, forced\_bos\_token\_id=tokenizer.lang\_code\_to\_id["en\_XX"])

translated\_text = tokenizer.batch\_decode(translated\_tokens, skip\_special\_tokens=True)[0]

return translated\_text

### Use of **@st.cache\_resource:**

In this **@st.cache\_resource** is used to avoid the reloading the translation model again and again what it will do is while starting the app it will load the model and the loaded model object will store in the cache memory as a model knowledge so it can able to translate without loading the model again and again.

### load\_translation\_model() Function:

This function is used to load the model and the tokenizer for the input and output text tokenizing is nothing but converting the input text in the numerical.

### def translate\_text\_local(text) Function:

In this the previous function will be called and respective results will stored in the model and tokenizer variable and by calling the function tokenizer and passing the arguments **text** - the **input given by the user** and **return\_Tensors - input** will be converted into the **tensor format** that the deep learning model can understand and final two thing is **max\_length** which specifies the **max allowed input tokens** and if the tokens is above **512** the **truncation** argument will **remove the remaining one**.

In line two the **\*\*inputs** used for the unpacking the dictionary that was send from the previous line in a tensor format and that input will be converted into the english text

translated\_text = tokenizer.batch\_decode(translated\_tokens, skip\_special\_tokens=True)[0]

This line is used to decode back the numbers into the word or letter format and if there are any special tokens like spaces or something it will ignore that and take the translated output and the output will be returned by the function.

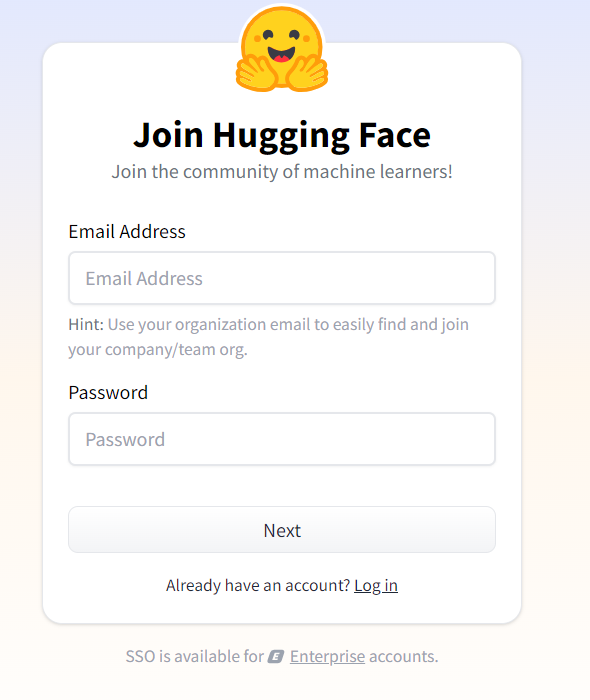
Other than this everything is similar to the previous(Version 1 ) code so I have not explained about it much.

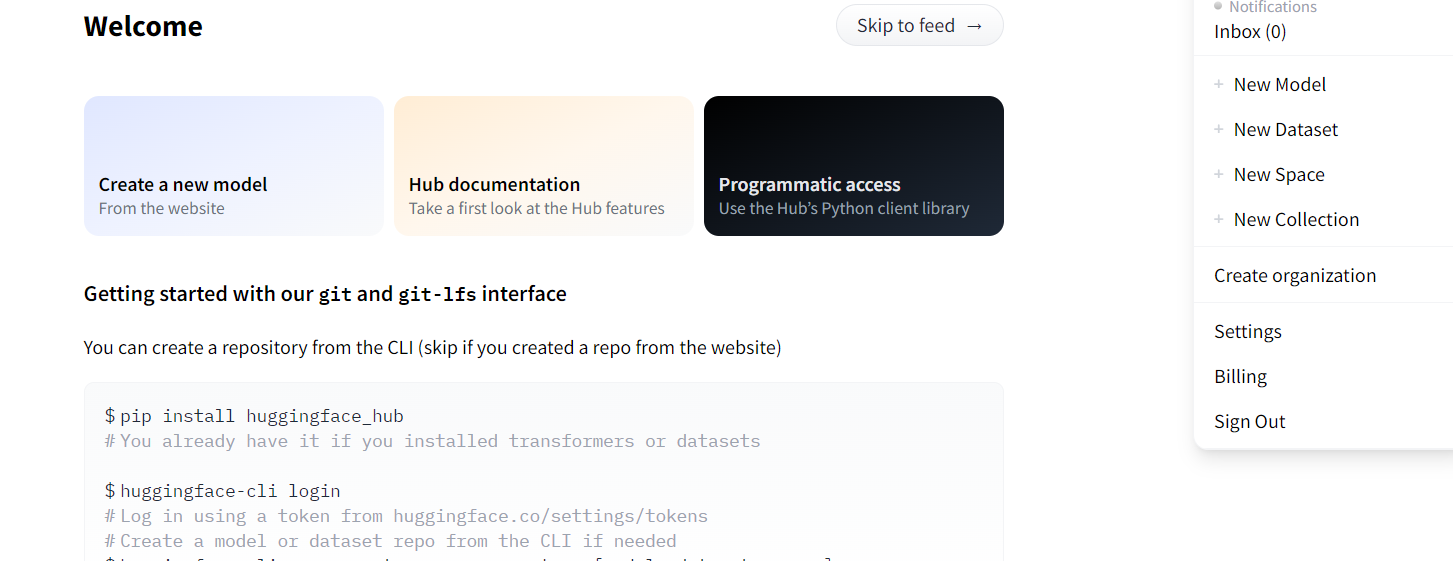
## Fully working App demo Video V2:

[Fusion\_MindVersion2.mp4](https://drive.google.com/file/d/164UsZKH25wqXLHstMifikk62CU176gW8/view?usp=sharing)

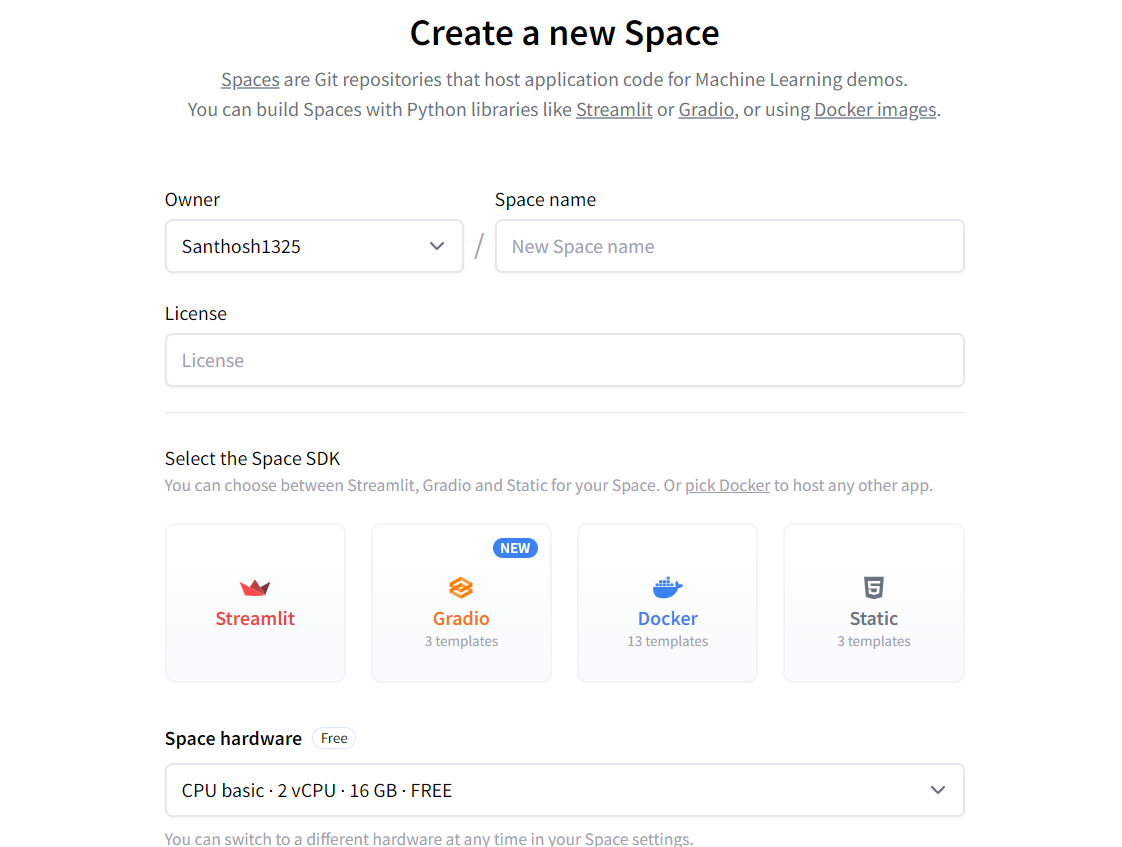
## Hugging Face setup process:

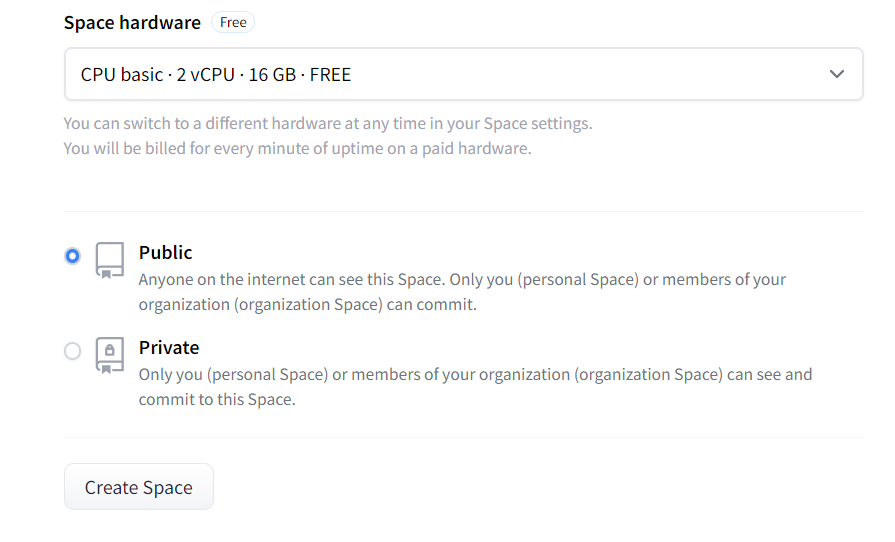
First need to create a hugging face account with the google account and then once the account is created means the need to set up with the some formality process such as Username , First name and Last name etc… and after all the process is done means there is a profile which has been created now will be appear in a top right corner of the screen when You clicked on the this tab will gets open profile means





In this click on the New space once you have clicked on the New space means you can able to create a space I the process of creating the spaces you need to give a space name and the app you want to create based on your preference after that need click the create new space.so the new space will be created.





This can also choose the specific hardware for the space to host and by clicking the public or private the app visibility will change.

For creating the secret key you need to go to the setting of your space and if you scroll down means you are able to find the new secret there you can create a new secret and you can store your API key which can be accessed using the getenv function.Finally the space is created and you can create a app and requirement file for creating the app and installing the required dependencies.

## App User Guide:

### How to Use This App:

#### Input Tamil Text:

You can either select one of the suggested Tamil phrases or input your own text. The app primarily focuses on Tamil inputs, but it supports a wide range of other languages as well (see the list below).

#### Generate Translations:

Once you've input your text, the app will automatically translate it to English. The translation model is a many-to-one model, meaning it can take input from various languages and translate it into English.

#### Generate Educational Content:

After translating the text into English, the app will generate educational content based on the translated input. You can adjust the creativity of the content generation using the temperature slider, and control the length of the output with the token limit setting.

#### Generate Images:

In addition to generating content, the app can also generate an image related to the translated content. You don’t need to worry about creating complex image prompts—FusionMind includes an automatic image prompt generator that will convert your input into a well-defined image prompt, ensuring better image generation results.

### 

### Features:

#### Multilingual Translation:

FusionMind supports a many-to-one translation model, so you can input text in a wide variety of languages, not just Tamil. Below are the supported languages:

Arabic (ar\_AR), Czech (cs\_CZ), German (de\_DE), English (en\_XX), Spanish (es\_XX), Estonian (et\_EE), Finnish (fi\_FI), French (fr\_XX), Gujarati (gu\_IN), Hindi (hi\_IN), Italian (it\_IT), Japanese (ja\_XX), Kazakh (kk\_KZ), Korean (ko\_KR), Lithuanian (lt\_LT), Latvian (lv\_LV), Burmese (my\_MM), Nepali (ne\_NP), Dutch (nl\_XX), Romanian (ro\_RO), Russian (ru\_RU), Sinhala (si\_LK), Turkish (tr\_TR), Vietnamese (vi\_VN), Chinese (zh\_CN), Afrikaans (af\_ZA), Azerbaijani (az\_AZ), Bengali (bn\_IN), Persian (fa\_IR), Hebrew (he\_IL), Croatian (hr\_HR), Indonesian (id\_ID), Georgian (ka\_GE), Khmer (km\_KH), Macedonian (mk\_MK), Malayalam (ml\_IN), Mongolian (mn\_MN), Marathi (mr\_IN), Polish (pl\_PL), Pashto (ps\_AF), Portuguese (pt\_XX), Swedish (sv\_SE), Swahili (sw\_KE), Tamil (ta\_IN), Telugu (te\_IN), Thai (th\_TH), Tagalog (tl\_XX), Ukrainian (uk\_UA), Urdu (ur\_PK), Xhosa (xh\_ZA), Galician (gl\_ES), Slovene (sl\_SI).

#### Temperature Adjustment:

You can adjust the temperature of the content generation. A higher temperature makes the content more creative and varied, while a lower temperature generates more focused and deterministic responses.

#### Token Limit:

Set the maximum number of tokens for content generation. This allows you to control the length of the generated educational content.

#### Auto-Generated Image Prompts:

One of the unique features of FusionMind is the auto-generated image prompts. Even if you're not experienced in creating detailed prompts for image generation, the app will take care of this for you. It automatically converts the translated text or content into a well-defined prompt that produces more accurate and high-quality images.

**Enjoy the multimodal experience with FusionMind and explore its powerful translation, content generation, and image generation features!**