***SANSKRITSCRIBE***

**

***INTRODUCTION***

. Sanskrit is one of the oldest and most significant languages, holding immense cultural, historical, and spiritual value. However, due to its complexity and **limited accessibility**, understanding and translating Sanskrit texts remains a challenge. The **AI-Powered Sanskrit Translator** is designed to **bridge this gap by providing an intelligent and efficient translation system** that can **extract and translate Sanskrit texts into multiple modern languages**.

This system integrates **OCR (Tesseract) and AI-based translation (Google Translate API)** to extract Sanskrit text from **scanned documents, PDFs, and images**, translating them into user-selected languages. It features **real-time translation, document processing, downloadable translations, and an easy-to-use interface**. Built using **Python (Flask), HTML, and libraries like Poppler, PyPDF2, and SQLite**, this project ensures **accurate, accessible, and seamless Sanskrit translations**, making it a valuable tool for researchers, students, and scholars interested in Sanskrit literature

***Objective***

The **Sanskrit Translator** aims to **modernize and simplify Sanskrit translations**, making them accessible to a broader audience. The key objectives include:

**🔹 Sanskrit Text Recognition & OCR Integration:**

* Uses **Tesseract OCR** to **extract Sanskrit text from scanned images and PDFs**.
* Converts handwritten or printed Sanskrit text into **editable and searchable text**.

**🔹 AI-Powered Translation:**

* Uses **Google Translate API** to **translate Sanskrit text into multiple modern languages**.
* Ensures **accurate and context-aware translations**, preserving meaning and structure.

**🔹 User-Friendly Interface:**

* Simple and **interactive UI with attractive styling**.
* Allows users to **upload documents, input text manually, and download translations**.

**🔹 Document Translation Support:**

* Supports **PDF, DOCX, and image-based text extraction** for efficient translation.
* Uses **Poppler and PyPDF2** to process text from PDFs.

**🔹 Download & Copy Translations:**

* Users can **download translated texts as PDFs** or **copy text with one click**.

**🔹 Translation History & Storage:**

* Saves previous translations for **quick access and retrieval**.

**🔹 Secure & Efficient Processing:**

* Ensures **data security and optimized performance** for **fast** translations.

***Scope of the Project***

The **Sanskrit Translator** provides a **comprehensive language translation solution**, covering:

**✅ Multiple Input Methods for Sanskrit Text:**

* Supports **typed text input, scanned document uploads, and image-based text extraction**.

**✅ OCR-Based Sanskrit Text Extraction:**

* Uses **Tesseract OCR & Poppler** to extract **editable text from Sanskrit manuscripts and PDFs**.

**✅ Multilingual Translation Capabilities:**

* Translates Sanskrit into **multiple languages** using AI-powered translation.

**✅ User-Friendly & Engaging UI:**

* Provides an **interactive interface** with **colorful styling and smooth navigation**.

**✅ Download & Copy Features:**

* Users can **download translated Sanskrit texts as PDFs** or **copy translations easily**.

**✅ Translation Storage & History:**

* Saves **previous translations** for future reference and research purposes.

**✅ Cross-Platform Compatibility:**

* Works on **desktops, tablets, and mobile devices**, making Sanskrit texts **more accessible**.

***Current System***

Traditional Sanskrit translation tools have **several limitations**, making them **less effective**. These include:

**❌ Limited Digital Access to Sanskrit Texts:**

* Many Sanskrit texts are **available only in printed manuscripts** and are not digitized.

**❌ Lack of OCR for Sanskrit Documents:**

* Most tools do not support **OCR for Sanskrit**, requiring **manual transcription**.

**❌ Limited Language Support in Translation:**

* Existing translation tools **mainly focus on English or a few select languages**.

**❌ Inaccurate & Inconsistent Machine Translations:**

* Many Sanskrit translation tools **fail to handle complex grammatical structures** accurately.

**❌ No Support for Document Formats (PDF, DOCX, Images):**

* Most tools **do not support scanned manuscripts, PDFs, or images**, making translations difficult.

**❌ Lack of Download & Storage Features:**

* Users **cannot store or retrieve** previous translations for research or study.

**❌ Difficult & Non-User-Friendly Interfaces:**

* Most Sanskrit translation platforms **lack an intuitive UI**, making them **challenging to use**.

***Proposed System***

The **Sanskrit Translator** introduces an **improved, AI-powered translation system** with advanced features:

**🚀 Advanced Sanskrit OCR for Text Extraction:**

* Uses **Tesseract OCR** to **extract text from scanned Sanskrit manuscripts, PDFs, and images**.
* Uses **Poppler** to **process PDF files and extract Sanskrit content** before translation.

**🚀 AI-Powered Machine Translation:**

* Uses **Google Translate API** for **accurate, context-aware** translations.
* Provides **real-time translations** in multiple languages.

**🚀 Multi-Format Document Support:**

* Allows users to **upload and translate PDFs, DOCX, and images (PNG, JPG, etc.)**.
* Ensures **clean and structured output for translated documents**.

**🚀 User-Friendly & Interactive UI:**

* Offers **aesthetic design and smooth navigation** for an improved user experience.
* Provides **drag-and-drop upload functionality** for easy document selection.

**🚀 Download & Copy Translations:**

* Users can **download translations as PDF files** for offline use.
* One-click **copy feature** for quick text usage.

**🚀 Storage & Translation History:**

* Saves **previous translations for quick access in future sessions**.
* Uses **SQLite** for managing and storing user translation history.

**🚀 Cross-Platform Accessibility:**

* Works on **desktops, tablets, and mobile devices**, ensuring **global accessibility**.

**🚀 Optimized Security & Performance:**

* Uses **secure encryption** for translation processing.
* Designed for **fast and efficient translation processing**, even for **large Sanskrit texts**.

***Future Scope***

The **Sanskrit Translator** has immense potential for **future improvements and enhancements**:

**🔮 Speech-to-Text for Sanskrit:**

* Future updates can include **voice-based Sanskrit translation**.

**🔮 Offline Translation Mode:**

* Implementing **offline translation capabilities** for Sanskrit texts.

**🔮 Enhanced AI for Contextual Accuracy:**

* Using **deep learning models** to **improve Sanskrit translation accuracy**.

**🔮 Integration with Language Learning Platforms:**

* Adding **educational tools and quizzes** for Sanskrit learners.

**🔮 Mobile App for Sanskrit Translation:**

* Developing **an Android/iOS app** for **on-the-go translations**.

**🔮 Cloud-Based Storage for Translations:**

* Enabling **users to save translations in the cloud** for easy access.

**🔮 Integration with Research & Digital Archives:**

* Partnering with **universities and libraries to digitize Sanskrit texts**.

**🔮 API for Businesses & Institutions:**

* Providing **a Sanskrit translation API** for **integration into websites & research tools**.

***Software & Tools Used***

The **Sanskrit Translator** is built using **Python, Flask, and HTML**, along with key libraries:

**📌 Backend Development (Python & Flask):**

* **Flask** – Manages the translation logic & document processing.
* **Google Translate API** – Provides AI-powered translation.

**📌 OCR & PDF Processing:**

* **Tesseract OCR** – Extracts text from scanned images & PDFs.
* **Poppler** – Converts PDFs to text-readable format.
* **PyPDF2** – Reads and processes PDF documents.

**📌 Frontend Development:**

* **HTML, CSS, JavaScript** – Enhances the UI for a better user experience.

**📌 Database & Storage:**

* **SQLite** – Stores translation history for easy retrieval.

**📌 Additional Libraries:**

* **ReportLab** – Generates **downloadable PDF translations**.

***SOFTWARE***

Device name Gameon

Processor 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz

Installed RAM 16.0 GB (15.6 GB usable)

Device ID 244AA89F-3C2F-4D62-901D-3F0BFBB6F359

Product ID 00342-43365-13608-AAOEM

System type 64-bit operating system, x64-based processor

***INTRODUCTION***

**SanskritScribe –**

***NURTURING YOUR CONNECTION TO ANCIENT WISDOM***

"Sanskritscribe- expertise in transcribing or writing texts in Sanskrit.

- Sanskrit, an ancient Indo-Aryan language, holds significant importance in various spiritual and philosophical traditions such as Hinduism, Buddhism, and Jainism.

- Scribe refers to copying and safeguarding religious scriptures, philosophical treatises, and historical records.

- Therefore, a "Sanskritscribe” is preservation, transcription, and dissemination of texts composed in Sanskrit, contributing to the perpetuation of its cultural and intellectual heritage.

Language plays a fundamental role in communication, cultural preservation, and knowledge sharing. Sanskrit, one of the world's oldest languages, is deeply rooted in historical texts, scriptures, and classical literature. As a language of immense significance in Indian traditions, philosophy, and religious texts, Sanskrit has been studied for centuries. However, with the rise of modern languages and digitalization, Sanskrit has become less accessible to the global audience. The need to translate Sanskrit into widely spoken languages has grown significantly, especially for scholars, researchers, and enthusiasts interested in understanding ancient texts.

AI-driven language translation technology has revolutionized the way languages are translated and understood. With advancements in Natural Language Processing (NLP), Optical Character Recognition (OCR), and machine learning, automatic translation of Sanskrit into multiple languages has become possible. The Sanskrit Translator project is an AI-powered system designed to translate Sanskrit text, documents, and scanned images into different languages with high accuracy and efficiency.

Unlike traditional translation methods, which require human expertise and extensive linguistic knowledge, AI-based solutions offer a fast and efficient way to bridge the gap between Sanskrit and other languages. The Sanskrit Translator project aims to break language barriers by providing seamless translations while preserving the meaning, syntax, and context of the original text.

The importance of Sanskrit translation extends beyond academics. Various fields, including philosophy, Ayurveda, yoga, and literature, require accurate translations of Sanskrit texts for better comprehension and research. The AI-powered translator will serve as a valuable tool for students, educators, historians, and professionals working with Sanskrit manuscripts and documents.

By integrating modern AI techniques with traditional language expertise, the Sanskrit Translator ensures that ancient knowledge is accessible to a broader audience. This project aims to contribute to the digital preservation of Sanskrit texts, enabling users to engage with this rich linguistic heritage effortlessly.

As technology continues to evolve, AI-driven translation models will improve in accuracy, making Sanskrit translations more refined and reliable. This project is a step forward in harnessing AI's power to promote linguistic and cultural heritage preservation.

***Motivation***

Sanskrit holds a prestigious position as the classical language of India, with its vast repository of texts covering philosophy, science, literature, and religious teachings. Despite its historical and cultural significance, Sanskrit has seen a decline in daily usage, making it challenging for modern learners and scholars to access and understand its content. This gap between Sanskrit texts and contemporary audiences motivated the development of an AI-based Sanskrit Translator.

The primary motivation behind this project is to bridge the linguistic divide and make Sanskrit texts comprehensible to non-native speakers. Many ancient scriptures and manuscripts contain invaluable knowledge, but their complexity and lack of translations hinder accessibility. By leveraging AI and NLP, the Sanskrit Translator provides an efficient means of converting Sanskrit content into widely spoken languages, ensuring the preservation and dissemination of knowledge.

Another driving factor is the increasing demand for Sanskrit translations in academic research. Scholars worldwide rely on translated Sanskrit texts for their studies in history, linguistics, philosophy, and spirituality. A robust AI-based translation system will aid researchers in obtaining precise and contextually accurate translations, enhancing their studies and interpretations.

In the educational domain, students and teachers often struggle with Sanskrit translations due to the intricate grammar and syntax of the language. The AI-powered translator simplifies this process, offering real-time translations that aid in learning and comprehension. This accessibility promotes greater interest in Sanskrit studies and encourages the preservation of linguistic traditions.

Religious texts in Sanskrit, such as the Vedas, Upanishads, and Puranas, hold deep spiritual and philosophical significance. Translating these texts into modern languages enables individuals to explore their meanings and teachings, fostering cultural and spiritual enrichment. The Sanskrit Translator project aims to make these texts widely available to people across the world.

With globalization and digital advancements, there is a growing trend towards integrating ancient knowledge with modern applications. Fields such as Ayurveda, yoga, and meditation rely on Sanskrit literature for foundational knowledge. By translating these texts, practitioners and researchers can gain deeper insights into their studies, ensuring that traditional wisdom continues to benefit society.

Finally, the project is motivated by the necessity of preserving endangered languages. As fewer people learn Sanskrit, the risk of losing significant portions of linguistic and cultural history increases. This AI-powered translator acts as a digital preservation tool, ensuring that Sanskrit remains accessible for generations to come.

***Scope of Project***

* **Text Translation**

**The project supports automatic translation of Sanskrit text into multiple languages, making it easier for users to understand and interpret ancient literature and scriptures. The translation is designed to maintain contextual accuracy and linguistic integrity, ensuring that users receive meaningful and precise outputs.**

* **OCR-Based Translation**

**The system integrates Optical Character Recognition (OCR) to extract and translate Sanskrit text from scanned images and handwritten documents. This functionality is beneficial for digitizing and translating old manuscripts, books, and inscriptions that are not available in digital format.**

* **Multi-Language Support**

**The translator provides output in multiple languages, ensuring accessibility for a wide range of users worldwide. Whether users need translations in English, Hindi, or other major languages, the system caters to diverse linguistic needs.**

* **Download & Copy Features**

**To enhance usability, the project includes options to download translated text as a document or copy it directly for further use. This feature is essential for researchers, educators, and professionals who need to store and reference translated content conveniently.**

* **User-Friendly Interface**

**The platform is designed with an intuitive and interactive UI, making it accessible for users of all backgrounds, including students, teachers, and scholars. The interface ensures smooth navigation and efficient interaction with the translation features.**

* **Integration with AI & NLP**

**The project utilizes AI-based NLP techniques to ensure accurate and contextually meaningful translations. By leveraging machine learning models trained on Sanskrit datasets, the system continuously improves its translation quality over time, making it a reliable tool for linguistic analysis.**

***Environment Used***

**Visual Studio Code (VS Code)**

**Visual Studio Code (VS Code) is a powerful, lightweight, and versatile code editor developed by Microsoft. It is widely used for software development due to its rich feature set, flexibility, and support for multiple programming languages, including Python, which is essential for this project. VS Code provides an efficient and seamless development environment that enhances productivity through intelligent code completion, debugging tools, and extensive extension support.**

**For the Sanskrit Translator Project, VS Code serves as the primary development environment, allowing developers to write, execute, and test Python code efficiently. Given the complexity of natural language processing (NLP) and optical character recognition (OCR) tasks involved in this project, VS Code’s integrated tools and extensions significantly aid in the development and debugging processes. The smooth integration with Flask, Tesseract OCR, and translation APIs ensures that the project's workflow remains organized and efficient.**

**Additionally, VS Code’s user-friendly interface, high customization options, and real-time collaborative features make it a preferred choice among developers working on AI-powered language translation applications. The editor supports real-time execution, making it easier to analyze results and improve model accuracy during testing and deployment.**

**Benefits of Using VS Code**

**1. Lightweight & Fast**

**One of the standout features of VS Code is its lightweight architecture, which ensures that it runs smoothly even on systems with limited resources. Unlike traditional Integrated Development Environments (IDEs) that can be resource-intensive, VS Code provides a responsive and efficient coding experience without consuming excessive memory or CPU power. This makes it particularly useful for handling large-scale projects such as AI-powered translation systems, where multiple dependencies and libraries are involved.**

**2. Integrated Debugging**

**VS Code comes with built-in debugging capabilities that make error detection and resolution much easier. Developers can set breakpoints, inspect variables, and step through code execution in real time, all within the editor. This feature is crucial for the Sanskrit Translator Project, as it allows for efficient debugging of OCR integration, API calls, and text translation functions. The debugging tools help in quickly identifying issues related to text extraction and language processing, improving overall accuracy and efficiency.**

**3. Extensive Extensions & Plugin Support**

**VS Code provides an extensive marketplace of extensions that enhance its functionality. For this project, various essential extensions are utilized, including:**

* **Python Extension – Provides syntax highlighting, debugging, and intelligent code suggestions.**
* **Flask Extension – Facilitates Flask development by providing route management and debugging tools.**
* **Jupyter Notebook Extension – Useful for testing and analyzing NLP models in an interactive environment.**
* **Tesseract OCR Extension – Helps in testing and improving OCR-based text extraction.**
* **Google Translate API Integration – Supports real-time translation by connecting the project with Google's translation services.**

**These extensions streamline the workflow, reducing development time and improving efficiency in implementing AI-driven text translation.**

**4. Built-in Terminal**

**VS Code features an integrated terminal that eliminates the need for external command-line tools. This built-in terminal allows developers to:**

* **Run Python scripts directly within the editor.**
* **Install and manage libraries using pip.**
* **Execute Flask applications without switching between windows.**
* **Perform OCR-related preprocessing tasks and text extractions.**
* **Debug API responses and server requests efficiently.**

**Having a terminal within the same development environment improves workflow efficiency, making it easier to manage dependencies and test different modules of the Sanskrit Translator Project.**

**5. Version Control Integration**

**VS Code seamlessly integrates with Git and GitHub, enabling efficient version control and collaborative development. With built-in Git features, developers can:**

* **Track changes in their code.**
* **Commit updates and push modifications to remote repositories.**
* **Collaborate with team members in real time.**
* **Roll back to previous versions if any issues arise during development.**

**For a project that involves AI-based text processing, version control is essential to maintain different versions of models, translations, and OCR processing techniques. This ensures that the best-performing version is retained and improvements can be made iteratively.**

**6. Intelligent Code Suggestions & Auto-Completion**

**VS Code is equipped with AI-powered IntelliSense, which provides real-time code suggestions and autocompletions. This feature is particularly beneficial in:**

* **Reducing development time by predicting and suggesting correct syntax.**
* **Identifying potential errors before execution.**
* **Helping developers quickly recall function names, parameters, and variable declarations.**

**For a project involving Natural Language Processing (NLP) and AI, where complex algorithms and multiple libraries are used, intelligent code suggestions enhance coding efficiency and accuracy.**

**7. Cross-Platform Support**

**VS Code is available on Windows, macOS, and Linux, making it accessible to developers across different operating systems. This flexibility ensures that the Sanskrit Translator Project can be developed and tested in various environments without compatibility issues. Additionally, developers can synchronize their settings and extensions across multiple devices, ensuring a consistent development experience.**

***Libraries Used***

The AI-Multilingual Translator project utilizes various Python libraries to implement key functionalities, including text translation, OCR, document handling, and web framework integration. Below is a detailed explanation of each library used in the project:

1. Flask – Flask is a lightweight web framework used to build the backend of the application. It allows easy integration of APIs and supports rendering web pages using HTML templates. Flask enables the seamless connection between the frontend and backend components.

* It allows developers to **create routes** (/translate, /upload, etc.) for different functionalities of the translator.
* Provides a **built-in development server** and debugger to test and troubleshoot web applications.
* Supports **Jinja2 templating**, which helps in rendering HTML pages dynamically based on user input.
* Enables **API integration** to connect with services like Google Translate for multilingual translation.
* Has built-in support for **request handling** (GET and POST requests) to process text, file uploads, and translation requests.

1. Googletrans – This library serves as an interface for Google Translate API, enabling automatic translation of text into multiple languages. It plays a crucial role in processing user input and returning translated results in real-time.

* Takes **user input (text or extracted text from OCR)** and translates it into the selected language.
* Returns translated output instantly and displays it in the UI.
* Handles **multiple languages**, allowing users to choose their desired output language.

1. Pytesseract – This Python wrapper for Tesseract OCR is used to extract text from images and scanned documents. It allows the system to recognize characters in different languages and convert them into editable text.

* Converts **image-based text** into editable digital text.
* Supports multiple languages, making it suitable for a multilingual translator.
* Can process **various file formats**, including JPG, PNG, and scanned PDFs.
* Works with OpenCV and Pillow to enhance text recognition accuracy.

1. pdf2image – This library converts PDF documents into images, enabling text extraction from scanned or digital PDFs. It is essential for handling document-based translations.

* PDFs often contain **scanned text** that cannot be extracted directly.
* Converts each PDF page into an image format (**PNG, JPG, etc.**) for OCR processing.
* Works alongside **PyMuPDF** to handle both **scanned and text-based PDFs**.

1. OS – The os module in Python provides functions to interact with the operating system, such as handling file paths, reading environment variables, and managing file directories. In this project, it helps with file handling for document uploads and processing.**Main Python File: app.py**

* Manages **file paths and directories** for storing uploaded documents.
* Reads and writes files during **text extraction and translation**.
* Helps with **document handling** (e.g., retrieving PDFs, images).

1. PIL (Python Image Library) - PIL, or Python Imaging Library, is essential for image processing tasks in SanskritScribe, providing essential functionality for manipulating and preprocessing images before OCR tasks. Its comprehensive features ensure compatibility and quality in image processing, enabling accurate extraction of Sanskrit text from uploaded images. By leveraging PIL, SanskritScribe enhances its capabilities, enabling users to seamlessly upload and process images containing Sanskrit text. This empowers users to engage with Sanskrit literature effectively, promoting a deeper understanding and appreciation of the language and its cultural heritage
2. **indic\_transliteration.sanscript**

**The indic\_transliteration library is a powerful tool for transliterating Sanskrit text between different scripts. In this project, it is used to convert Sanskrit text written in one script (such as Devanagari) into another script (such as IAST or Romanized format).**

* **Transliteration Support – Converts Sanskrit text from Devanagari (संस्कृतम्) to other scripts like IAST (saṃskṛtam), Romanized, and more.**
* **Improves Readability – Makes Sanskrit content accessible to users unfamiliar with Devanagari script.**
* **Enhances NLP Processing – Some machine learning and NLP models perform better when Sanskrit is in Romanized form (IAST) rather than in Devanagari.**
* **Helps in Phonetic Representation – Users can better understand pronunciation when Sanskrit text is transliterated into IAST (International Alphabet of Sanskrit Transliteration).**

***Main Python File: app.py***

**The app.py file is the core backend component of the Sanskrit Translator Project. It serves as the central processing unit for handling various translation-related tasks, including text input, transliteration, text and document translation, OCR-based text extraction, and response generation. The file is developed using Python and Flask, integrating machine learning (ML), natural language processing (NLP), optical character recognition (OCR), and API-based translation techniques to deliver accurate and efficient translations of Sanskrit text into multiple languages.**

**This file acts as a bridge between the user and the translation models, receiving user input (text or document), processing it, and returning a translated output in the desired language. Additionally, it ensures seamless communication between the frontend interface and backend logic, making the translation process smooth and interactive.**

**Libraries Used in app.py**

1. **Flask** – Used for backend development and handling web requests.
2. **Googletrans** – Manages language translation functions.
3. **Pytesseract** – Extracts text from images for OCR-based translation.
4. **pdf2image & PyMuPDF** – Extracts text from PDFs and converts them for translation.
5. **OS** – Used for file handling and system interactions.
6. PIL (Python Library Image): PIL preprocesses images for OCR tasks in SanskritScribe, ensuring accuracy.
7. indic\_transliteration.sanscript is a Python library that enables script conversion (transliteration) of Indian languages, allowing seamless transformation between **Devanagari, IAST, and other scripts**.

***API Used in This Project***

The **Sanskrit Translator Project** primarily relies on the **Google Translate API** and the **Indic Transliteration API** for translation and transliteration functionalities. These APIs work together to provide **accurate, efficient, and meaningful translations** of Sanskrit text into multiple languages while preserving grammatical structure and linguistic nuances.

**1. Google Translate API**

The **Google Translate API** is one of the most powerful machine translation services, enabling real-time conversion of text between various languages. It supports **neural machine translation (NMT)**, which enhances the accuracy of translations by understanding the context rather than performing mere word-to-word conversions.

* **Why Google Translate API?**
  + It offers support for **over 100 languages**, ensuring that Sanskrit translations are available in widely spoken languages such as English, Hindi, German, and French.
  + Uses **deep learning algorithms** to improve translation quality over time, reducing errors and increasing fluency.
  + Provides a **RESTful API**, which allows seamless integration with Flask-based applications, making it easy to send and receive translated text via HTTP requests.
  + Supports **automatic language detection**, which is useful when users input Sanskrit text in different scripts (like Devanagari, IAST, or Romanized Sanskrit).
* **How It Works in Our Project?**
  + When a user enters Sanskrit text for translation, the **app.py** file sends an API request to Google Translate.
  + The API processes the text using **neural translation models** and returns the translated output.
  + The translated text is displayed on the UI, with options to **copy, download, or further refine** the translation.
  + This process ensures that the system delivers high-quality translations while handling complex Sanskrit linguistic structures.

**2. Indic Transliteration API (Sanscript Module from indic\_transliteration.sanscript)**

The **Indic Transliteration API**, specifically from the indic\_transliteration.sanscript module, plays a crucial role in converting Sanskrit text **between different scripts**. Sanskrit can be written in various scripts, including **Devanagari, IAST (International Alphabet of Sanskrit Transliteration), and Romanized Sanskrit**.

* **Why Indic Transliteration API?**
  + Enables **seamless transliteration** of Sanskrit text into different scripts, making it accessible to scholars, students, and researchers who use different writing systems.
  + Supports **multiple transliteration schemes**, including **Devanagari, IAST, ISO, and Harvard-Kyoto**, ensuring maximum compatibility.
  + Preserves the **original phonetics** of Sanskrit words, ensuring that the transliterated output is **accurate and readable**.
  + Essential for **OCR-based translations**, where scanned Sanskrit texts might need conversion into a readable and processable format before translation.
* **How It Works in Our Project?**
  + When a user enters Sanskrit text in **one script**, the indic\_transliteration.sanscript module converts it into the desired target script.
  + The converted text is then passed to the **Google Translate API** for further translation into different languages.
  + This ensures that users from various linguistic backgrounds can input Sanskrit text in their preferred script and receive an accurate translation.

***CODING PART***

**MAIN FILE – SANSKRIT TRANSLATOR**

To open the source file, we will first open the visual studio code .

**OPENING VISUAL STUDIO CODE :**

**STEP 1 :**

In your windows search bar , type visual studio code

**STEP 2 :** Click on the open and it will open the visual studio code.

A screenshot of a computer

AI-generated content may be incorrect. The interface after opening looks like this –

**STEP 3:**

Opening the project directory which is –

"C:\Users\amaan\OneDrive\Desktop\Sanskrit Translator" in visual studio code by selecting open folder and then

A screenshot of a computer

AI-generated content may be incorrect.

It will be seen like this:

A screen shot of a computer program

AI-generated content may be incorrect.

STEP 4: In this folder, the subdirectories and files looks like this:

A screenshot of a computer program

AI-generated content may be incorrect.

Let us begin with code explanation:

***PYTHON***

Uploaded imagePython is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum and first released in 1991, Python emphasizes code readability with its clean and straightforward syntax, making it an excellent choice for beginners and experienced developers alike. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python's extensive standard library and a vast ecosystem of third-party packages enable its use in a wide range of applications, from web development and data analysis to artificial intelligence, scientific computing, and automation. Its cross-platform compatibility and active community further contribute to its popularity as one of the most widely used programming languages in the world.

This line of code is part of a Python web application built using the Flask framework. It imports essential components from the flask library: Flask is the main class used to create the web application instance; render\_template is used to render HTML templates; request is used to handle data sent from the client side (such as form inputs or file uploads); jsonify is used to return JSON responses from the server; and send\_file is used to send files from the server to the user, such as downloaded documents or translated output. These imports provide the foundational functionality needed to create interactive, user-friendly web applications.



The line import os in Python is used to include the built-in os (Operating System) module, which provides a way to interact with the operating system in a platform-independent manner. This module allows the program to perform various tasks such as accessing environment variables, creating or deleting files and directories, changing file paths, and executing system commands. In web or desktop applications, os is often used to manage file paths, handle file uploads or downloads, and navigate directories dynamically, making the application more flexible and adaptable to different environments.



The line import pytesseract is used to include the pytesseract module in a Python program, which is a Python wrapper for Google's Tesseract-OCR (Optical Character Recognition) engine. It allows the program to extract text from images by recognizing the characters and converting them into editable digital text. This is particularly useful in projects that involve reading printed or handwritten text from scanned documents, images, or PDFs. By using pytesseract, developers can build powerful OCR-based applications like text translators, document scanners, and automated data entry systems with ease and accuracy.



The line from PIL import Image is used to import the Image class from the Python Imaging Library (PIL), now maintained as the Pillow library. This class enables the program to open, manipulate, and process image files in various formats such as JPEG, PNG, BMP, etc. In OCR and translation projects, Image is commonly used to load and handle image files before passing them to tools like pytesseract for text extraction. It provides essential image handling functions such as resizing, cropping, converting formats, and displaying images, making it a core component in image processing workflows in Python.



The line from pdf2image import convert\_from\_path imports the convert\_from\_path function from the pdf2image library, which is used to convert PDF documents into image formats (usually JPEG or PNG). This is especially useful in projects where you need to extract text from PDFs using OCR tools like pytesseract, as OCR works on images rather than directly on PDF files. With this function, each page of a PDF can be converted into a separate image, making it possible to perform text recognition or translation on the content of PDF documents seamlessly.

The line from docx import Document imports the Document class from the python-docx library, which is used to create, read, and modify Microsoft Word (.docx) files in Python. This class allows you to programmatically generate professional Word documents by adding text, headings, tables, images, and formatting styles. In the context of a translator project, this can be helpful for reading content from a Word file for translation or exporting translated text into a new, downloadable .docx document for the user.

The line from googletrans import Translator imports the Translator class from the googletrans library, which is a Python wrapper for Google Translate. This class enables automatic language translation of text from one language to another using Google’s translation engine. In a multilingual translator project, it plays a key role in translating the extracted or user-inputted text into the desired language selected by the user, making the translation process simple, efficient, and accurate.



The line from indic\_transliteration.sanscript import transliterate, DEVANAGARI, IAST imports essential components from the indic\_transliteration library, which is used for converting text between different Indian language scripts. Here, transliterate is a function that performs the actual script conversion, while DEVANAGARI and IAST are constants representing two common script formats—Devanagari (used for Sanskrit, Hindi, etc.) and IAST (International Alphabet of Sanskrit Transliteration). This is particularly useful in a Sanskrit translation project where you may want to transliterate Sanskrit text from Devanagari to IAST or vice versa to improve readability and accessibility for different users.



A black background with colorful text

AI-generated content may be incorrect.The line app = Flask(\_\_name\_\_) is used to create an instance of the Flask application. Here, Flask is a class from the Flask web framework, and \_\_name\_\_ is a special variable in Python that represents the name of the current module. By passing \_\_name\_\_ to the Flask class, it helps Flask determine the root path of the application, which is necessary for locating templates, static files, and other resources.

This line initializes the web application and serves as the starting point for defining routes and handling user requests.

* 1. **Operating System Check:**
* if os.name == "nt":  
  This line checks if the operating system is Windows.
  + "nt" stands for **New Technology**, which is used by Windows-based systems.
  + If the system is Windows, the block inside this if statement will be executed.
  1. **Setting the Tesseract Executable Path:**
* pytesseract.pytesseract.tesseract\_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
  + This line specifies the **absolute path to the Tesseract-OCR executable file**.
  + It is required because pytesseract (a Python wrapper for Tesseract OCR) needs to know **where the Tesseract engine is installed** on the system.
  + This is especially important on Windows, where the system doesn't automatically detect the Tesseract path.
  1. **Setting the Poppler Path:**
* POPPLER\_PATH = r"C:\Users\amaan\Downloads\Release-24.08.0-0.zip"
  + Poppler is a tool required to **convert PDF files into images**, which is used by libraries like pdf2image.
  + This line sets the path to the **Poppler for Windows** zip file (it should ideally be extracted, and the path should point to the extracted folder containing pdftoppm.exe).
  + It's essential for enabling **PDF to Image conversion**, especially when handling scanned PDF documents.

A screen shot of text

AI-generated content may be incorrect.

**Function Definition:**

* def extract\_text\_from\_image(image\_path):
  + This line defines a **function named extract\_text\_from\_image** which takes one parameter: image\_path.
  + The parameter image\_path is the **file path of the image** from which text is to be extracted.

**Opening the Image:**

* image = Image.open(image\_path)
  + Uses the PIL.Image module to **open the image file** from the given path.
  + This step loads the image so that it can be processed by the OCR engine.

**Extracting Text Using OCR:**

* text = pytesseract.image\_to\_string(image, lang="san")
  + This line uses the pytesseract library to **perform Optical Character Recognition (OCR)** on the image.
  + The parameter lang="san" tells Tesseract to recognize **Sanskrit language text** in the image.
  + Tesseract tries to identify and extract any readable text present in the image.

**Stripping Extra Spaces:**

* return text.strip()
  + The .strip() function is used to **remove any unwanted whitespace or newline characters** from the beginning and end of the extracted text.
  + It then **returns the clean text** to the caller.

**Error Handling with Try-Except:**

* except Exception as e:
  + This block handles any errors that might occur during image loading or text extraction.
  + If an error occurs, it returns a message like **"Error extracting text from image: [error message]"** which helps in debugging.

This function is designed to **automatically extract Sanskrit text from an image file** using Tesseract OCR. It handles potential errors gracefully and returns either the cleaned extracted text or an error message. It's a crucial part of the translator project, especially for handling image-based document inputs.

A computer screen shot of text

AI-generated content may be incorrect.

**Function Definition:**

* def extract\_text\_from\_pdf(pdf\_path):
* This line defines a **function named extract\_text\_from\_pdf**, which takes one parameter: pdf\_path.
* The parameter pdf\_path refers to the **file path of the PDF document** that needs to be processed.

**Convert PDF to Images:**

* images = convert\_from\_path(pdf\_path)
* This line uses the **convert\_from\_path() function from the pdf2image library** to convert each page of the PDF into separate image files.
* This step is essential because Tesseract OCR works on images, not directly on PDF files.

**Initialize Empty Text String:**

* extracted\_text = ""
* An empty string is initialized to **store the extracted text** from each page of the PDF.

**Extract Text from Each Page (Image):**

* for img in images:
  + A loop is started to **iterate through all the image pages** converted from the PDF.
* extracted\_text += pytesseract.image\_to\_string(img, lang="san") + "\n"
  + For each image, **Tesseract OCR is applied to extract Sanskrit text** using lang="san".
  + The extracted text is **appended to the extracted\_text variable**, and a newline (\n) is added for readability between pages.

**Return Cleaned Extracted Text:**

* return extracted\_text.strip()
  + This line **removes extra whitespace or newline characters** from the beginning and end of the full extracted text and **returns the final result**.

**Error Handling:**

* except Exception as e:
  + If there is any **issue in PDF conversion or OCR processing**, this block will handle the error.
  + It returns a **formatted error message** like: "Error extracting text from PDF: [error message]", which helps in debugging.

This function is designed to **extract Sanskrit text from a PDF file**. It first converts the PDF pages into images, then uses **Tesseract OCR** to extract text from each page. It supports multiple-page PDFs and handles errors effectively. This function plays a vital role in the project by enabling **PDF document translation** capabilities.

A screen shot of a computer code

AI-generated content may be incorrect.

**Function Definition:**

* def extract\_text\_from\_docx(docx\_path):
* This line defines a **function named extract\_text\_from\_docx** which takes one parameter: docx\_path — the file path of the DOCX file to be processed.

**Open the DOCX File:**

* doc = Document(docx\_path)
* This line uses the **Document class from the python-docx library** to open and load the DOCX file content.

**Extract Paragraph Text:**

* "\n".join([para.text for para in doc.paragraphs])
* This line creates a list of **text from each paragraph** in the DOCX file using a list comprehension: [para.text for para in doc.paragraphs].
* All paragraph texts are then **joined together with newline characters (\n)**, forming a complete readable text.

**Return Extracted Text:**

* The function returns the **combined paragraph text**, providing a clear textual representation of the DOCX file.

**Error Handling:**

* except Exception as e:
  + If any **error occurs while reading the DOCX file or extracting the text**, this block catches it.
  + return f"Error extracting text from DOCX: {str(e)}" returns a helpful error message that includes the error details.

This function efficiently reads a DOCX file and extracts all its paragraph content in a clean and readable format. It ensures the program doesn't crash on error and gives appropriate feedback. This function is a key part of your project as it allows **Sanskrit** A computer screen shot of text

AI-generated content may be incorrect.**document (.docx) translation and text extraction**.

**Function Definition:**

* def translate\_text(text, dest\_language):
* This defines a function named translate\_text
* which takes two parameters:
  + text: The text that needs to be translated.
  + dest\_language: The **target language code** (e.g., 'en' for English, 'hi' for Hindi, 'sa' for Sanskrit, etc.).

**Initialize Translator:**

* translator = Translator()
* This line initializes an instance of the Translator class from the googletrans module, used for translating text to various languages via the **Google Translate API**.

**Check for Sanskrit Language (Special Case):**

* if dest\_language == "sa":
  + Sanskrit translation is handled differently.
* return transliterate(text, IAST, DEVANAGARI)
  + Here, **transliteration** is used instead of traditional translation.
  + It converts **text written in Latin/IAST (International Alphabet of Sanskrit Transliteration)** into **Devanagari script**, which is used for Sanskrit.

**Standard Translation for Other Languages:**

* If the destination language is **not Sanskrit**, the code proceeds with standard translation using:
  + translated = translator.translate(text, dest=dest\_language)
  + This line uses the Google Translate API to convert the text into the desired language.
* return translated.text returns only the translated string.

**Error Handling:**

* except Exception as e: catches any runtime errors (e.g., internet issues or API failure).
* return f"Translation error: {str(e)}" provides a readable error message to the user.

This function serves as the **core translation mechanism** in your project. It supports multiple languages and includes **special handling for Sanskrit** using transliteration. It’s robust, handles errors gracefully, and ensures smooth multilingual translation functionality in your AI-powered translator project.

A black background with white text and colorful lines

AI-generated content may be incorrect.

The line translated\_text = "" is a **simple but important line** that helps store and manage translated content globally within your Flask-based translator application.

A screen shot of a computer screen

AI-generated content may be incorrect.

This code is a part of a **Flask web application** and is responsible for **defining the homepage (root route)** of the website. The @app.route('/') is a **decorator** that tells Flask that when a user accesses the root URL (/) of the web application (i.e., http://localhost:5000/), it should trigger the index() function. Inside this function, render\_template('index.html') is called, which means Flask will **search for an HTML file named index.html in the templates folder** and render (display) it in the user's browser.

This route acts as the **starting point or landing page** of the translator application, where users will see the main interface to interact with—such as uploading files, entering text, or initiating translation. In essence, this block of code connects the **backend Python logic to the frontend HTML page**, forming the foundation A computer screen with colorful text

AI-generated content may be incorrect.for user interaction in the application.

This code defines a **Flask route /translate** which handles **POST requests**—typically sent when the user submits a form to translate a file or some input text. The @app.route('/translate', methods=['POST']) decorator indicates that this function will be executed whenever a POST request is made to the /translate URL. Inside the translate() function, the variable translated\_text is declared as **global**, meaning it can be accessed and modified across different functions in the script. It is reset to an empty string at the start to clear any previous translations.

Then, the function checks whether a **file has been uploaded by the user** using 'file' in request.files and ensures the filename is not empty (request.files['file'].filename). If a file is indeed uploaded, it is extracted using request.files['file'] and assigned to the variable file. The **file extension** is determined by splitting the filename at the dot (.) and converting it to lowercase using file.filename.split('.')[-1].lower() to help identify its type (e.g., pdf, docx, png). Finally, file\_path = f"uploads/{file.filename}" constructs the path where the uploaded file will be temporarily stored in the **uploads directory** for further processing like OCR A black background with multicolored text

AI-generated content may be incorrect.and translation.

This section of the code is responsible for **saving the uploaded file** from the user into a local folder called "uploads" on the server. First, os.makedirs("uploads", exist\_ok=True) ensures that the uploads directory exists—if it doesn’t, it creates one. The parameter exist\_ok=True prevents the program from throwing an error if the folder already exists, making the process smooth and error-free. Once the directory is ensured, the uploaded file (received via the form) is saved to the specified location using file.save(file\_path). Here, file\_path is the full path where the file should be stored (e.g., uploads/document.pdf). This saved file will later be used for **text extraction and translation processing** in the application.

A screen shot of a computer program

AI-generated content may be incorrect.

This section is responsible for **determining the type of uploaded file** and **extracting text accordingly** using the appropriate method.

The code checks the file extension using file\_ext:

* If the file is an **image** (.jpg, .jpeg, .png, .webp), it calls extract\_text\_from\_image() to extract text using OCR.
* If it's a **PDF**, it uses extract\_text\_from\_pdf() to convert PDF pages to images and then applies OCR.
* If it's a **DOCX document**, it uses extract\_text\_from\_docx() to read the text directly from the Word file.
* If the format is not supported, it returns an error response using jsonify.

After the text extraction is done, the uploaded file is deleted from the server using os.remove(file\_path) to keep the server clean and optimize storage.

If no file is uploaded, this else block allows users to **input text manually via a form** instead of uploading a document. It fetches that text from the form using request.form.get().

Finally, if no text was extracted either from the file or the form, it returns an error message indicating **no text was found**, ensuring proper validation and user feedback.

A screen shot of a computer code

AI-generated content may be incorrect.

* **request.form.get("language", "en")**:
  + This line retrieves the **target language** selected by the user from the form submission.
  + If the user does **not select any language**, it defaults to **English ("en")**.
  + Example: If the user chooses **Hindi**, the value would be "hi"; for **Sanskrit**, it could be "sa".
* **translate\_text(extracted\_text, target\_language)**:
  + This line **calls the translate\_text() function**, passing the extracted text and the desired target language.
  + Inside this function, if the target language is **Sanskrit**, it applies **transliteration** from Latin to Devanagari.
  + For other languages, it uses the **Google Translate API** to perform the translation.

return jsonify({"text": translated\_text})

This is especially useful when your app is communicating with JavaScript on the frontend, making it easy to display the output dynamically without reloading the page.

A black screen with white text

AI-generated content may be incorrect.

The given code defines a Flask route @app.route('/download'), which handles the **download functionality** for the translated text in the application. Inside the associated function download\_translation(), it first checks whether any **translated text is currently available** by referring to the global variable translated\_text. If this variable is **empty or not populated**, it means that no text has been translated yet, and the function responds with an error message **"No translation available"** along with an **HTTP status code 400**, indicating a bad request. This check ensures that the user does not attempt to download a file when there is **nothing to download**, thus avoiding unnecessary errors or confusion.

A black screen with white text

AI-generated content may be incorrect.

This block of code is responsible for **creating and sending the translated text file to the user for download**. The line file\_path = "translated\_text.txt" defines the filename for the output file. Using a **with open()** statement, it opens (or creates) the file in **write mode ("w") with UTF-8 encoding** to ensure compatibility with multiple languages, especially those using special characters like Sanskrit. The translated content stored in the global variable translated\_text is then **written into this file**. After successfully writing the content, the function uses **send\_file()** to send the file back to the client/browser as an **attachment**, triggering a download prompt. This allows users to **save the translated content as a text file** on their system.

A black background with white text

AI-generated content may be incorrect.

This code block is the **entry point of the Flask application**. The condition if \_\_name\_\_ == '\_\_main\_\_': ensures that the Flask app will run only when the script is executed **directly**, not when it's imported as a module in another file. Inside this block, app.run(debug=True) starts the **Flask development server** and enables **debug mode**, which is very useful during development. With debug mode enabled, the server will **automatically reload on code changes** and also provide **detailed error messages** if something goes wrong, helping developers identify and fix issues more easily.

***HTML [HyperText Markup Language], CSS***

In today's globalized world, language barriers pose a significant challenge in communication. To overcome this, we have developed an **AI-Powered Multilingual Translator**, a web-based application that enables users to translate text and documents into multiple languages efficiently. This project integrates **HTML, CSS, and JavaScript** with backend support for translation services, making it a powerful and user-friendly tool.

***Technology Stack***

* **HTML**: Structures the webpage and provides input fields for text and file translation.
* **CSS**: Enhances the UI with a modern and responsive design, including gradient backgrounds, interactive buttons, and well-styled text areas.
* **JavaScript**: Implements the translation logic, handles user interactions, and communicates with the backend for text processing.
* **Backend (Flask/Python)**: Processes user inputs, utilizes **OCR for scanned documents**, and connects with the **Google Translate API** for multilingual translations.

***Key Features***

✅ **Manual Text Translation** – Users can input text and select the target language for instant translation.  
✅ **File Upload & Translation** – Supports document translation, including scanned images (OCR-enabled).  
✅ **Copy & Download Translations** – Users can copy the translated text or download it for later use.  
✅ **User-Friendly Interface** – Designed with an intuitive layout, attractive colors, and seamless user interaction.

This project aims to provide a **fast, reliable, and accessible** translation tool for individuals, businesses, and educational institutions, bridging the language gap efficiently.

A computer screen with text

AI-generated content may be incorrect.

This HTML code defines the **basic structure and layout** of the front-end interface for the **Sanskrit Translator web application**. It begins with the <!DOCTYPE html> declaration, indicating an HTML5 document. The <head> section includes essential metadata such as character encoding (UTF-8) and viewport settings for responsive design, ensuring the webpage is **mobile-friendly** and scales properly on all screen sizes. The title of the web page is set as **“Sanskrit Translator”**, which appears in the browser tab.

Additionally, the code integrates **Bootstrap 5.3.0** using a CDN link to provide a modern, responsive, and styled user interface without writing custom CSS from scratch. In the <body>, a light grey background color is applied using inline CSS (background-color: #f8f9fa;) to enhance readability and visual comfort.

A <div> with classes container mt-5 p-4 bg-white shadow rounded creates a centered, white box with padding, margin from the top, a shadow effect, and rounded corners — giving it a clean and professional appearance. Inside this container, an <h2> header displays the title **“📜 Sanskrit Translator”** in the center with some margin at the bottom, serving as the main heading of the web interface and setting the tone for the application’s purpose.

A screen shot of a computer code

AI-generated content may be incorrect.

This section of the HTML code creates the **user input area** for the Sanskrit Translator web application. It consists of two major components — **Text Input Section** and **File Upload Section**, both enclosed in <div> elements with the mb-3 Bootstrap class for margin spacing.

1. **Text Input Section:**  
   The first <div> displays a label prompting the user to *"Enter text in any language:"* using the <label> tag with the class form-label for Bootstrap styling. Below the label, a <textarea> element is provided with the id="text-input" and class form-control, which gives it a responsive and styled appearance. The rows="3" attribute controls the visible height of the text area, and the placeholder="Type here..." gives a hint to the user before they begin typing. This section allows users to **manually type or paste content** that they want to translate.
2. **File Upload Section:**  
   The next <div> presents another labeled input asking users to *"Upload a file (PDF, DOCX, Image):"*. An <input> element of type file is used here, with id="file-upload" and styled with form-control. This enables users to **upload documents or images** containing text for automatic OCR and translation. Supported formats are **PDF, DOCX, JPG, PNG**, etc., as stated in the label, ensuring flexibility in how the content is provided to the application.

Together, these sections make the interface **user-friendly and versatile**, offering both **manual text input and file-based text extraction options**.

A screen shot of a computer code

AI-generated content may be incorrect.

This section of the HTML code is responsible for the **Language Selection** functionality of the Sanskrit Translator web application. It is designed using Bootstrap classes to ensure a clean and user-friendly layout.

1. The outer <div> with the class mb-3 provides vertical spacing below this section for neat alignment.
2. Inside this <div>, a <label> with the class form-label is displayed, prompting the user with the text *"Translate to:"*. This instructs the user to select the target language in which they want their input text or uploaded content to be translated.
3. Below the label, a <select> element with the id="language-select" and class form-select is used. This element creates a **dropdown menu** that lets users choose their **preferred output language** from a list of available options.
4. The <option> tags inside the <select> element define the available languages. The values assigned to each option (like "sa" for Sanskrit, "en" for English, "hi" for Hindi, "fr" for French, and "es" for Spanish) are used programmatically by the backend to determine the selected language for translation.

Overall, this section enables the user to **customize the translation output language**, making the application multilingual and accessible to a wider audience.

A screen shot of a computer code

AI-generated content may be incorrect.

This section of the HTML code handles the **interaction and output display** part of the Sanskrit Translator web application. It contains three main components: the **Translate button**, **translated text output**, and the **Download button**, all styled using Bootstrap for a clean and intuitive user interface.

1. The **Translate Button** is placed inside a <div> with the class text-center to center-align it. The button itself is styled using the class btn btn-primary, giving it a prominent blue appearance. The onclick="translateText()" attribute binds this button to a JavaScript function called translateText(), which is responsible for sending the user input (text or uploaded file) to the server for translation.
2. The **Translated Text Output** section is used to display the result returned by the server. It starts with a <h4> heading titled *"Translated Text:"*, followed by a <p> tag with the ID translated-text. This paragraph is initially empty but gets dynamically populated with the translated content once the user clicks the Translate button. It has styling classes like border, p-3, bg-light, and rounded to make it visually distinct and readable.
3. Lastly, the **Download Button** allows users to download the translated text in a file format. It’s again center-aligned using text-center, with a green-themed button styled using btn btn-success. The onclick="downloadTranslation()" event handler triggers another JavaScript function that communicates with the backend to download the translated content as a file.

Together, these elements provide a seamless user experience by facilitating input processing, real-time output display, and easy access to translated content.

A screen shot of a computer code

AI-generated content may be incorrect.

This script block defines the beginning of the **translateText() function**, which is triggered when the user clicks the **"Translate"** button on the web page. The function is responsible for collecting the user’s input data and preparing it for sending to the Flask backend for processing.

1. First, it **retrieves the manually typed text** from the textarea using document.getElementById("text-input").value. This allows users to input text directly into the browser.
2. Next, it **accesses any uploaded file** (such as an image, PDF, or DOCX file) using document.getElementById("file-upload").files[0]. If a file is selected by the user, it will be included in the translation request.
3. Then, it **captures the selected language** for translation from the dropdown menu (<select> element) using document.getElementById("language-select").value. This value is sent to the server to determine the target language (e.g., Sanskrit, English, Hindi, etc.).

In short, this part of the function is gathering all necessary inputs (text, file, and language selection) before sending them to the server for translation. The rest of the function (which likely follows this snippet) would handle sending the data using an AJAX request (such as fetch()), receiving the response, and displaying the translated result on the webpage.

A screen shot of a computer program

AI-generated content may be incorrect.

This segment of the translateText() function handles how user inputs (text or file) and the selected language are packaged and sent to the backend server for processing. A **FormData object** is created using let formData = new FormData();—this object allows for easily collecting and sending form fields, including files, in a format that the server can interpret.

Next, the code checks whether a file is uploaded by the user. If a file exists (if (file)), it appends the file to the FormData object using formData.append("file", file);. If no file is selected, it instead appends the manually typed text with formData.append("text", text);. This ensures that either text or a file is sent, depending on the user's input method.

Then, it appends the **selected target language** to the form data using formData.append("language", language);, so that the server knows which language to translate into.

Finally, the fetch() API is used to send an **asynchronous HTTP POST request** to the Flask backend at the /translate route. The formData is included in the request body, allowing the server to receive and process the user input for translation. The remaining portion of the function (not shown here) would typically handle receiving the server's response and displaying the translated text on the webpage.

A black background with text

AI-generated content may be incorrect.

This part of the translateText() function is responsible for **handling the server response** after the translation request is sent. Once the fetch() function makes a POST request to the /translate route, it uses .then(response => response.json()) to convert the server’s response into a JSON object, which is easier to work with in JavaScript. The next .then(data => { ... }) block processes that JSON data.

If the response contains a text property (i.e., a successful translation result), it dynamically updates the web page by inserting the translated text into the HTML element with the ID translated-text using innerText. If the response doesn't include the text (meaning there was likely an error during processing), the code displays an error message instead, using either the specific data.error provided by the server or a default message like "Unknown error occurred."

Additionally, if something goes wrong in the fetch operation itself (like server being unreachable or response format being invalid), the .catch(error => { ... }) block catches that exception and shows an error message with error.message, ensuring that users are always given feedback even if a problem occurs. This provides a **robust and user-friendly error handling mechanism** for a smoother user experience.

A screen shot of a computer

AI-generated content may be incorrect.

This section defines the **downloadTranslation() function**, which is triggered when a user clicks the **"Download Translation"** button. The function makes a **fetch request to the /download route** on the server. This route is designed to return the translated text as a downloadable file.

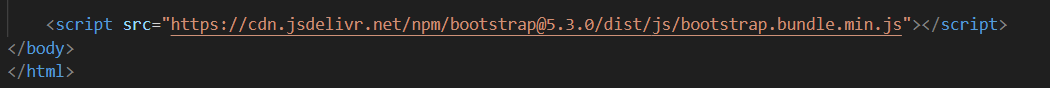
The fetch("/download") sends a **GET request**, and once the server responds, the code checks whether the response was successful using response.ok. If the response is valid (i.e., the server has provided a translated file), it proceeds to retrieve the file content using response.blob(). A **blob (binary large object)** represents the file’s raw data, which can then be used for creating a downloadable link on the browser.

If the response is not okay—perhaps because there is no translation available yet—the code throws an error with the message "No translation available for download." This ensures that users are informed when there's an issue with the download process, maintaining a **clear and user-friendly interface**.

A computer screen shot of a program code

AI-generated content may be incorrect.

This part completes the **downloadTranslation() function**, which handles the process of enabling users to download the translated text as a .txt file. Once the **server responds with a blob (a file-like object)**, the function dynamically creates an **<a> (anchor) element** using document.createElement("a"). The blob is converted into a **temporary downloadable URL** using window.URL.createObjectURL(blob) and assigned to the href of the anchor element. Then, the download attribute is set to "translated\_text.txt", which defines the default filename for the downloaded file. The line a.click(); **simulates a click event**, triggering the file download automatically without needing user interaction with the anchor element itself. If any error occurs during this process (like a network issue or missing file), the .catch() block catches the error and displays an **alert box with the error message**, ensuring the user is informed of any problems during the download process. This approach provides a smooth and interactive experience for downloading translated text directly from the web page.



The last part of the HTML page includes the **Bootstrap JavaScript Bundle via CDN** using the <script> tag. The source URL "https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js" ensures that Bootstrap’s interactive components—like modals, dropdowns, carousels, and tooltips—function properly on the webpage. The **bootstrap.bundle.min.js file** includes both **Bootstrap's core JavaScript** and **Popper.js**, which is required for dynamic UI behaviors (such as positioning tooltips or dropdown menus). Placing this script just before the closing </body> tag is a best practice in web development, as it allows the HTML content to load first, improving the page's performance and load time. The </body> and </html> tags then close the body and the entire HTML document respectively, marking the end of the webpage structure.

***JavaScript***

**JavaScript** is a high-level, interpreted programming language that is primarily used to make web pages interactive. It is a **client-side scripting language**, meaning it runs in the user’s browser, enabling functionalities like animations, form validations, event handling, and API interactions. JavaScript is one of the core technologies of web development, alongside **HTML and CSS**.

JavaScript plays a crucial role in our **AI-Powered Multilingual Translator** project as it enables interactivity and dynamic content updates without needing to reload the webpage. Specifically, we use JavaScript to:

✅ **Handle User Inputs** – It captures text input, file uploads, and selected languages for translation.  
✅ **Send Requests to the Backend** – Using the fetch() API, JavaScript sends text or files to the backend (Flask/Python) for processing.  
✅ **Update Translated Text Instantly** – Once the translation is received from the backend, JavaScript updates the webpage dynamically to display the translated output.  
✅ **Enhance User Experience** – It ensures a smooth and interactive experience by handling button clicks, form submissions, and displaying results without page refresh.

A screenshot of a computer program

AI-generated content may be incorrect.

The translateText() function is designed to handle the translation process when a user interacts with the "Translate" button on the webpage. Inside this function, the first line retrieves the **user-inputted text** from the <textarea> element using document.getElementById("text-input").value. The second line checks whether the user has uploaded any file (such as an image, PDF, or DOCX) by accessing the first file from the file input field file-upload. The third line captures the **selected target language** from the dropdown menu (language-select). These three inputs—manual text, uploaded file (if any), and chosen language—are then used to prepare the translation request that will be sent to the server via a form submission using JavaScript's fetch() API (in the remaining part of the function).

A computer screen with text

AI-generated content may be incorrect.

1. **Creating Form Data:**
   * The FormData object is initialized using let formData = new FormData();.
   * This object is used to store key-value pairs that will be sent to the server in a structured format.
2. **Handling File Uploads:**
   * If a **file** is selected (if (file)), it gets added to the FormData object using formData.append("file", file);.
   * Additionally, the selected **target language** (from the dropdown) is also appended (formData.append("language", language);).
   * This ensures that the server knows which language the uploaded file’s text should be translated into.
3. **Handling Manual Text Input:**
   * If no file is uploaded (else block), the manually entered text (text) is added to FormData using formData.append("text", text);.
   * The target language is **forced to Sanskrit (san)** using formData.append("language", "san");.
   * This means that any manually entered text will always be translated into **Sanskrit**, while uploaded files can be translated into any selected language.

This logic ensures that the correct translation request is sent based on whether the input is a **file** or **typed text**.

A computer code on a black background

AI-generated content may be incorrect.

This section of the JavaScript function plays a crucial role in sending the user’s input (either a typed text or an uploaded file) to the server for translation and then displaying the translated output on the web page. The fetch() method is used to make a POST request to the server route /translate, along with the form data (formData) which includes either the manually typed text or the uploaded file, along with the target language. Once the request is successfully sent, the server processes it, performs the translation, and sends back a response.

The response from the server is then converted into a JSON object using response.json(). This allows JavaScript to easily access the translated text returned by the server. The next part of the code takes this translated text and inserts it into the HTML element with the ID translated-text, thereby updating the webpage dynamically without needing to reload it. This seamless communication between frontend and backend ensures a smooth user experience, where users can see their translated output immediately after clicking the "Translate" button.

A screen shot of a computer code

AI-generated content may be incorrect.

This JavaScript function downloadText() is designed to allow users to download the translated text directly from the web page in the form of a .txt file. Here's how it works, explained in a paragraph:

When the function is triggered (typically by clicking a "Download" button), it first retrieves the content of the translated text from the HTML element with the ID translated-text. This is the area where the translated output is displayed on the webpage. It then creates a **Blob** object—a data object representing the translated text in plain text format (text/plain). After that, the function dynamically creates an <a> (anchor) HTML element, assigns it a download link that points to the blob object (using URL.createObjectURL()), and sets the download attribute to specify the filename (translated\_text.txt). Finally, the function programmatically clicks this link using link.click(), which initiates the download process in the browser without needing to reload or redirect the page. This approach ensures users can instantly save their translated content as a file with a single click, enhancing usability and convenience.

After applyning codes in vs code.. Open terminal and type-

Python app.py

A black background with white text

AI-generated content may be incorrect.

Click on the <http://127.0.0.1:5000> to run Sanskrit translator Interface

A screen shot of a computer

AI-generated content may be incorrect.

SANSKRIT TRANSLATOR looks like this:

A screenshot of a computer

AI-generated content may be incorrect.

“So here we’re going to test out SANSKRIT TRANSLATOR”

Example 1: Enter text in any language in text column like {My Name Is Janhavi Singh}.

A screenshot of a computer

AI-generated content may be incorrect.

Then press translate, the translator change this letters to sanskrit-

A screenshot of a computer

AI-generated content may be incorrect.

After that you’ll also download the translated texts for your future reference.

Example 2: Upload images which contains sanskrit text like this-

A close up of a book

AI-generated content may be incorrect.

First browse the image from the pc-

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Then we translate the texts of the image-

A screenshot of a computer

AI-generated content may be incorrect.

The texts are successfully transtlated-

After that you’ll download the translated information for thefuture refence by pressing the download button-

A screenshot of a computer

AI-generated content may be incorrect.

The information directily download in the pc in downloads section-

A black screen with a black stripe

AI-generated content may be incorrect.

In my Sanskrit translator website you’ll translate sanskrit texts in different languages like- English, Hindi, French, Spenish and also you’ll translate these languages to sanskrit-

A screenshot of a computer

AI-generated content may be incorrect.

Example 3: Here we’re translating written docx into different languages-

First choose the file for translation-

A screenshot of a computer

AI-generated content may be incorrect.

After choosing the file translate it in your required language, I select english language-

A screenshot of a computer

AI-generated content may be incorrect.

After pressing translate, the document will translate the sanskrit text into english like this

A screenshot of a computer

AI-generated content may be incorrect.

The texts are successfully translated, you’ll download it for your future refrence by pressing download translation.

***DOWNLOADS***

How my translations download in my pc-

The **Sanskrit Translator Project** not only offers real-time translation of user-inputted text or uploaded documents but also provides a convenient feature to **download the translated content as a file**. This feature is especially helpful for users who wish to keep translated text for **future reference, offline use, academic work, or documentation purposes**.

The **Download functionality** is implemented using: ✅ **Flask (Backend Processing)** – To generate and serve the translated file.  
✅ **JavaScript (Frontend Interaction)** – To handle download events.  
✅ **HTML (User Interface)** – To provide a clickable download button for users.

**1️. How the Download Feature Works**

Once the user completes a translation, the system stores the translated output and provides a **Download** option. When clicked, the translated content is delivered to the user’s computer as a .txt file.

**👉 Step-by-step Process:**

1️⃣ The user inputs text or uploads a file and selects the target language (e.g., Sanskrit).  
2️⃣ The **Flask backend processes** the text and returns the translated result.  
3️⃣ The translated content is **stored as a text file** (e.g., translated.txt) on the server.  
4️⃣ The **Download button becomes available** on the interface.  
5️⃣ When the user clicks **Download**, a request is sent to the Flask **/download route**.  
6️⃣ Flask responds by **sending the translated file to the browser as a downloadable attachment**.  
7️⃣ The file is automatically saved in the **Downloads folder** of the user’s device.

**2️. Functions That Enable the Download Feature**

The download process is enabled using both backend and frontend components.

**🅐 Flask Route for Download Handling**

* A special route (/download) is created in the Flask app.
* This route is responsible for **sending the translated file to the browser**.

**Key Concepts:** 🔹 send\_file() – Sends the file from the server to the user’s browser.  
🔹 os.path.join() – Locates the path of translated.txt.  
🔹 as\_attachment=True – Ensures the file is **downloaded instead of being opened** in the browser.

**What Happens Here?** ✔️ The user clicks Download → Flask receives a request → Locates translated.txt in the server folder → Sends it back → The file starts downloading on the user’s device.

**🅑 JavaScript Function to Initiate Download**

The **JavaScript function** on the frontend makes the download possible by connecting the browser to the Flask backend.

**How it works:** ✅ The function runs when the user clicks the **Download button**.  
✅ It automatically sends a request to the **/download route**, triggering file download.

**In Action:** 1️⃣ User clicks "Download Translation"  
2️⃣ JavaScript function triggers a fetch or redirects the browser to /download  
Browser receives file → File gets saved to PC

**🅒 HTML Button in the Interface**

The HTML button acts as the user’s tool to download the file.

**Working of the Button:** 🔹 Has an onclick attribute that calls the JavaScript function  
🔹 As soon as the user clicks, the download process starts  
🔹 The file is delivered without needing additional interaction

**3️. Where the File is Saved**

Once downloaded, the translated file appears in the **Downloads folder** of the user’s PC by default.

📄 **File Format:** .txt  
🛠️ **Can be opened using:**

* Notepad (Windows)
* TextEdit (Mac)
* VS Code, Sublime Text, Atom
* Microsoft Word or any text reader

**Use Cases of the Download Feature**

This download feature makes the translator highly practical and user-friendly for various needs:

🔸 **Students & Researchers** – Save translated papers, books, or notes.  
🔸 **Professionals** – Store translated business content or official letters.  
🔸 **Writers & Authors** – Maintain multiple versions of their work.  
🔸 **Language Learners** – Download and revise translated phrases and sentences.

***CONCLUSION***

The Sanskrit Translator project represents a significant technological advancement in making ancient Sanskrit literature accessible to modern users through the integration of AI and OCR technologies. By combining powerful tools such as Tesseract OCR for text recognition and Google Translate API for multilingual translation, this system bridges the gap between the ancient and digital worlds. The project successfully addresses the limitations of traditional translation tools by enabling users to translate scanned documents, images, PDFs, and manually entered text into multiple modern languages. Its user-friendly interface, multi-format document support, real-time translation output, and download capabilities enhance the overall user experience and make the system practical for students, researchers, scholars, and language enthusiasts alike.

Moreover, this project has not only digitized Sanskrit translation but also laid a strong foundation for future enhancements such as speech-to-text features, offline translation modes, mobile applications, and cloud integration. The ability to store translation history and download translated content ensures the tool’s utility beyond a one-time use, making it a valuable resource for repeated academic or personal reference. With its secure, fast, and efficient processing, the Sanskrit Translator stands as a comprehensive and forward-looking solution, contributing to the preservation, learning, and global dissemination of one of the world’s most classical languages in the digital era.

***LIMITATIONS***

* **Limited Accuracy of OCR for Complex Fonts and Poor Image Quality**  
  The accuracy of text extraction using OCR (Tesseract) heavily depends on the quality and clarity of the scanned documents or images. Handwritten texts, faded manuscripts, and documents with decorative or ancient fonts may not be recognized accurately, leading to incorrect translations.
* **Dependency on Internet for Translation API**  
  The system relies on third-party translation APIs like Google Translate, which require an active internet connection. In offline scenarios or restricted environments, the translation service becomes unavailable, limiting the project's usability.

* **Translation Quality for Sanskrit May Vary**  
  Sanskrit is a complex and context-sensitive language. Existing APIs may not always provide semantically accurate or grammatically correct translations, especially when translating idiomatic expressions, compound words (samasa), or verse-based texts.
* **Limited Language Pair Support for Sanskrit**  
  While common languages like English, Hindi, and Spanish are well-supported, some rare language pairs may not offer high-quality translations when Sanskrit is involved. This restricts the effectiveness of multilingual translation beyond commonly used languages.
* **No Text-to-Speech or Audio Translation Feature**  
  The project currently lacks a text-to-speech function that could help users listen to translated output. This could have benefited visually impaired users or those learning pronunciation in Sanskrit and target languages.
* **No Translation Context Customization or Manual Correction Tools**  
  Users cannot refine, edit, or suggest context-based translations manually. There is no built-in feedback system or dictionary enhancement feature to improve translation quality or learn user preferences over time.
* **Lack of Mobile Responsiveness or Application Version**  
  The current implementation is designed for desktop use via web browsers. It does not include a dedicated mobile application or responsive UI design, limiting accessibility for smartphone users or on-the-go translation requirements.

***FUTURE SCOPE***

1. **Integration of Advanced AI/ML-based Translation Models**  
   In future iterations, the system can be upgraded by incorporating deep learning-based Natural Language Processing (NLP) models specifically trained on Sanskrit corpora. These models can enhance translation accuracy by understanding complex grammar structures and context.
2. **Offline Translation Capability**  
   Adding offline translation features using pre-trained local language models or offline dictionaries will make the system more robust and accessible, especially in areas with limited or no internet connectivity.
3. **Text-to-Speech and Speech-to-Text Functionalities**  
   Implementing text-to-speech (TTS) and speech-to-text (STT) functionalities would enable users to listen to translations and provide voice input. This would make the system more interactive and accessible to users with visual or motor impairments.
4. **Mobile Application Development**  
   The development of a mobile version (Android/iOS) of the translator can significantly increase its reach and usability. A dedicated app would allow users to access translation services on-the-go and increase convenience.
5. **Addition of Grammar Correction and Contextual Editing Tools**  
   Enhancing the system with grammar correction tools and contextual editing features would allow users to manually refine translations, making the results more personalized and accurate for academic or official use.
6. **Support for Additional Document Formats and Multilingual PDFs**  
   Currently limited to basic files, the system can be extended to support DOCX, PPTX, XLSX, scanned multi-page PDFs, and even ZIP file batches. It could also handle multilingual documents and provide page-wise translations.
7. **Collaborative Learning and Dictionary Building**  
   A user-feedback-based learning system can be implemented where users contribute to improving translation accuracy by correcting outputs. A built-in collaborative Sanskrit dictionary or glossary could be developed to enrich the system over time.

***REFERENCES***

* **Google Translate API Documentation**  
  <https://cloud.google.com/translate/docs>  
  (Used for implementing real-time language translation functionalities.)
* **Tesseract OCR Engine – Optical Character Recognition Tool**  
  <https://github.com/tesseract-ocr/tesseract>  
  (Used for extracting text from scanned documents and image files.)
* **Flask – Lightweight Python Web Framework**  
  <https://flask.palletsprojects.com/>  
  (Used to build the backend server and manage translation requests.)
* **HTML, CSS, and JavaScript Documentation – W3Schools**  
  <https://www.w3schools.com/>  
  (Used for designing the frontend and integrating interactivity.)
* **Sanskrit Language Resources – Sanskrit Documents Collection**  
  <https://sanskritdocuments.org/>  
  (For understanding grammar rules and linguistic structure of Sanskrit.)
* **Bootstrap Framework – Frontend UI Library**  
  <https://getbootstrap.com/>  
  (Used to design responsive and professional UI components.)
* **Natural Language Toolkit (NLTK) – Python NLP Library**  
  <https://www.nltk.org/>  
  (For future scope: to implement grammar and linguistic features in Sanskrit translation.)
* **Research Articles on Sanskrit Language Digitization**  
  Various articles and whitepapers on digitization and computational linguistics applied to ancient Indian languages like Sanskrit.