The TechNovas
Group 52

Automated Step-By-Step Visual Performance Guide Generation from Sinhala Demonstration Videos

Supervisor: Dr. L. Ranathunga

Team Members

Kularathna M.D.S.A.

204104H

Nethmini S.A.R.

204137K

Dilshan K.G.A.P.

204041K

Introduction

- Many Sinhala demonstration videos lack performance guides, making it challenging for viewers to understand key details.
- Our project focuses on solving the above problem by generating visual performance guides from demonstration videos.
- Key domain areas include Natural Language Processing, Machine Learning and Image/Video Processing.

PROBLEM IN BRIEF

- Viewer Frustration: Users have to rewind and skip forward repeatedly to locate important details in the video.
- Accessibility Issues: Difficulty in quickly finding and understanding key information due to excessive filler content.
- No Extraction Tools : There are no options to extract key information from Sinhala demonstration videos.

AIM & OBJECTIVES

Aim

- To automatically generate visual performance guides from Sinhala demonstration videos, allowing viewers to quickly grasp key information through synchronized text and visuals.

Objectives

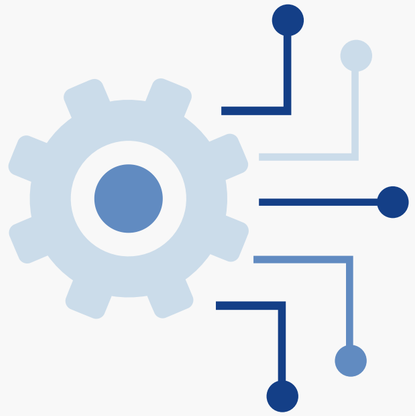
1. Convert Sinhala audio to text, correct it and generate meaningful content.
2. Identify key points and create step-by-step instructions.
3. Capture and enhance key visuals to focus on important steps.
4. Generate a document by combining relevant visuals and instructions.
5. Enhance text accuracy, image selection, and performance guide quality using machine learning techniques.
6. Evaluate and benchmark the success factor for the algorithms used for the above operations.

MODULES OF THE SYSTEM

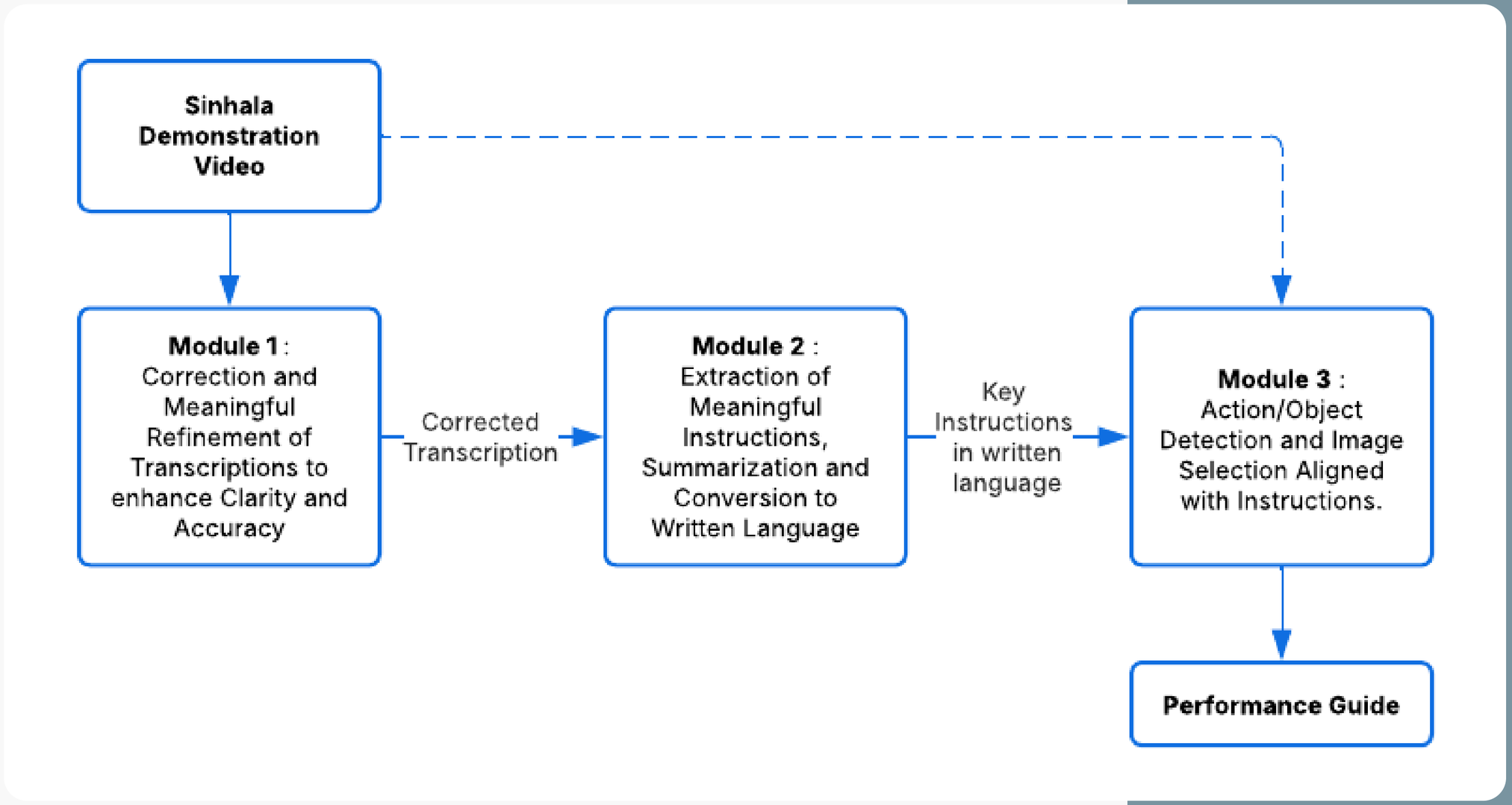
Module 1
Correction and Meaningful Refinement of Transcriptions to enhance Clarity and Accuracy

Module 2
Extraction of Meaningful Instructions, Summarization and Conversion to Written Language

Module 3
Action/Object Detection and Image Selection Aligned with Instructions.



High Level Architecture



Module 1 : Correction and Meaningful Refinement of Transcriptions to enhance Clarity and Accuracy

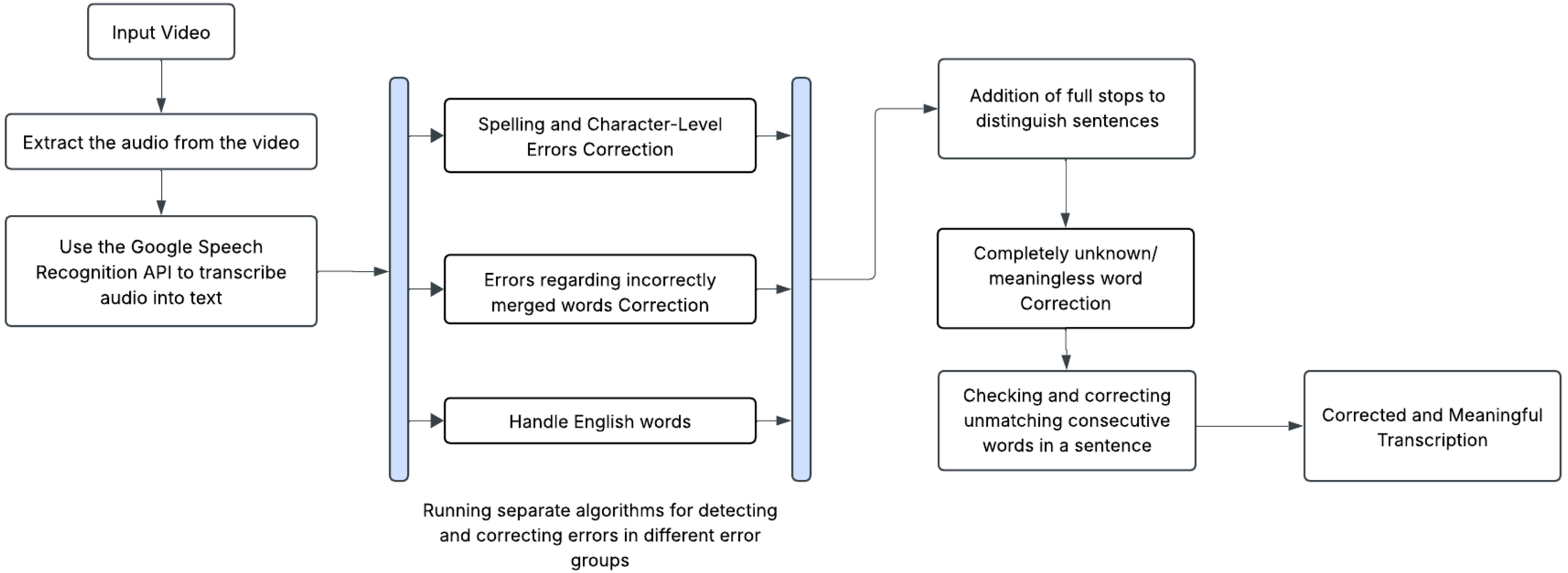
Introduction

- Focuses on transcribing audio and improving the understandability and accuracy of the transcribed text. There are 6 groups of errors which will be separately considered and fixed.

Challenges

- Sinhala speakers often merge words when speaking fast, making it difficult to distinguish individual words.
- Many Sinhala speakers use English words when talking in Sinhala.

Module 1 Diagram



Methodology

- **Misspelled words-**
  - Tokenize text
  - Check words against corpus to find mis-spelt words.
  - Use edit-distance calculation to find appropriate words to replace mis-spelt words.

- **Merged words-**
  - Tokenize text
  - Check with corpus to find invalid words which have more than 4 letters.
  - Invalid words are split using NLTK into 2 words. Different combinations can be formed.
  - Calculate the edit distance for each combination.
  - Combo with smallest edit distance from valid words will be the most accurate.

- **English words -**
  - Tokenization.
  - Find words that are English, with English characters
  - Transliterate these words

Methodology

- **Full stop -**
  - Using POS tagging with rule based approach

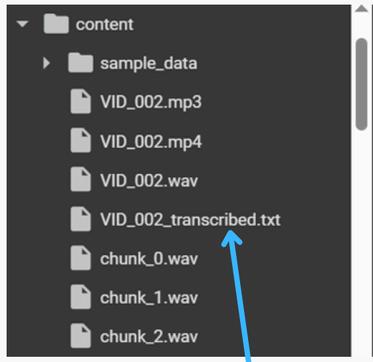
- **Unknown words -**
  - Sentence splitting (segmentation)
  - Compare with corpus to find remaining unavailable words.
  - Use model for masked language modeling
  - Train model with text from the cooking domain
  - Obtain appropriate word.

- **Check whether words make sense -**
  - Consider nouns, verbs and adjectives
  - Compare with top predicted words

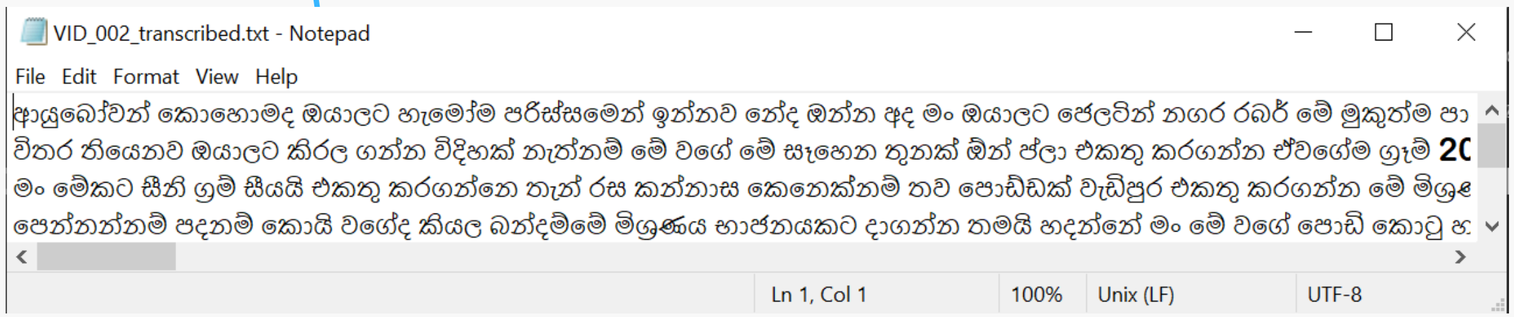
Implementation

Converting video to text file

```
!pip install SpeechRecognition pydub

import speech_recognition as sr
from pydub import AudioSegment

# Function to convert MP4 to MP3
def convert_mp4_to_mp3(input_file):
    video = AudioSegment.from_file(input_file, format="mp4")
    video.export("VID_002.mp3", format="mp3", bitrate="128k")
    print(f'Conversion complete! The MP3 file is saved.')

input_video = '/content/VID_002.mp4'
convert_mp4_to_mp3(input_video)

# Function to split audio into 5-minute chunks
def split_audio_into_chunks(audio_path, chunk_duration_ms=3*60*1000):
    audio = AudioSegment.from_wav(audio_path)
    chunks = []

    # Split the audio into chunks of the specified duration (5 minutes here)
    for start_ms in range(0, len(audio), chunk_duration_ms):
        chunk = audio[start_ms:start_ms+chunk_duration_ms]
        chunks.append(chunk)

    return chunks

# Function to convert audio to text in chunks
def sinhala_speech_to_text(audio_path):
    recognizer = sr.Recognizer()

    # Convert audio to WAV format if needed
    if not audio_path.endswith(".wav"):
        audio = AudioSegment.from_file(audio_path)
        audio_path = audio_path.rsplit(".", 1)[0] + ".wav"
        audio.export(audio_path, format="wav")

    # Split the audio into 5-minute chunks
    audio_chunks = split_audio_into_chunks(audio_path)

    full_text = ""

    for i, chunk in enumerate(audio_chunks):
        # Export the chunk to a temporary WAV file
        chunk_path = f"chunk_{i}.wav"
        chunk.export(chunk_path, format="wav")

        # Recognize speech in each chunk
        with sr.AudioFile(chunk_path) as source:
            print(f"Processing chunk {i + 1}...")
            audio_data = recognizer.record(source)

            try:
                text = recognizer.recognize_google(audio_data, language="si-LK")  # Sinhala language code
                full_text += text + "\n"
                print(f"Chunk {i + 1} Transcribed Text: {text}")
            except sr.UnknownValueError:
                print(f"Chunk {i + 1}: Google Speech Recognition could not understand the audio")
            except sr.RequestError:
                print(f"Chunk {i + 1}: Could not request results from Google Speech Recognition service")

    # Save the transcribed text to a file
    output_txt = audio_path.rsplit(".", 1)[0] + "_transcribed.txt"
    with open(output_txt, "w", encoding="utf-8") as f:
        f.write(full_text)

    print(f"Transcribed text saved as {output_txt}")

# Set the path manually
audioPath = "/content/VID_002.mp3"  # Change this to your file path

# Call the function with the specified path
sinhala_speech_to_text(audioPath)
```



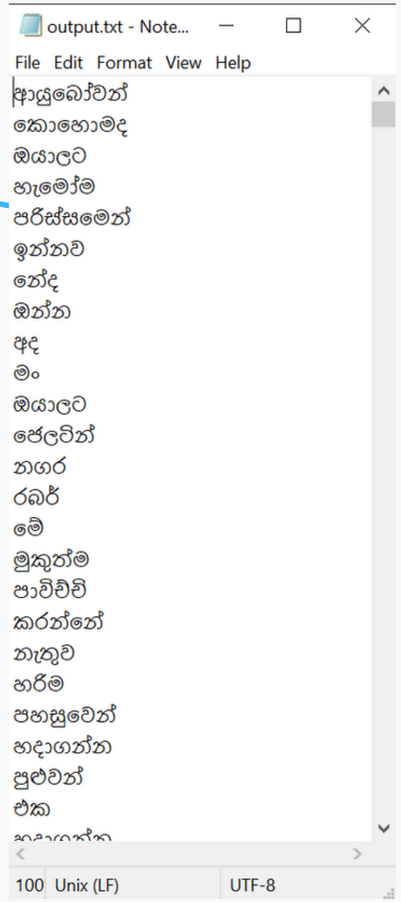

```
Conversion complete! The MP3 file is saved.
Processing chunk 1...
Chunk 1 Transcribed Text: ආයුබෝවන් කොහොමද ඔයාලට හැමෝම පරිස්සමෙන් ඉන්නව නේද ඔන්න අද මං ඔ
Processing chunk 2...
Chunk 2 Transcribed Text: මං මේකට සීනි ග්‍රෑම් සීයයි එකතු කරගන්නෙ තැන් රස කන්නාස කෙනෙක්නම් තව
Processing chunk 3...
Chunk 3 Transcribed Text: එක කපලත් ඛලමු ඇතුළ කොහොමද තියෙන්නේ කියල ඔයාලට බලාගන්න පුලුවන්
Transcribed text saved as /content/VID_002_transcribed.txt
```



Implementation

Tokenization


```
[5]  !pip install indic-nlp-library

     from indicnlp.tokenize import indic_tokenize

     # Define the path to your saved text file
     file_path = "/content/VID_002_transcribed.txt"

     # Read the file content
     with open(file_path, "r", encoding="utf-8") as f:
         sinhala_text = f.read()

     # Tokenize words
     tokens = indic_tokenize.trivial_tokenize(sinhala_text, lang='si')
     with open("/content/output.txt", "w", encoding="utf-8") as f:
         f.write("\n".join(tokens))

     with open("/content/output.txt", "r", encoding="utf-8") as f:
         sinhala = f.read()

     # Print tokenized and cleaned words
     print("Tokenized Words:", sinhala)
```



Module 2 : Extraction of Meaningful Instructions, Summarization and Conversion to Written Language

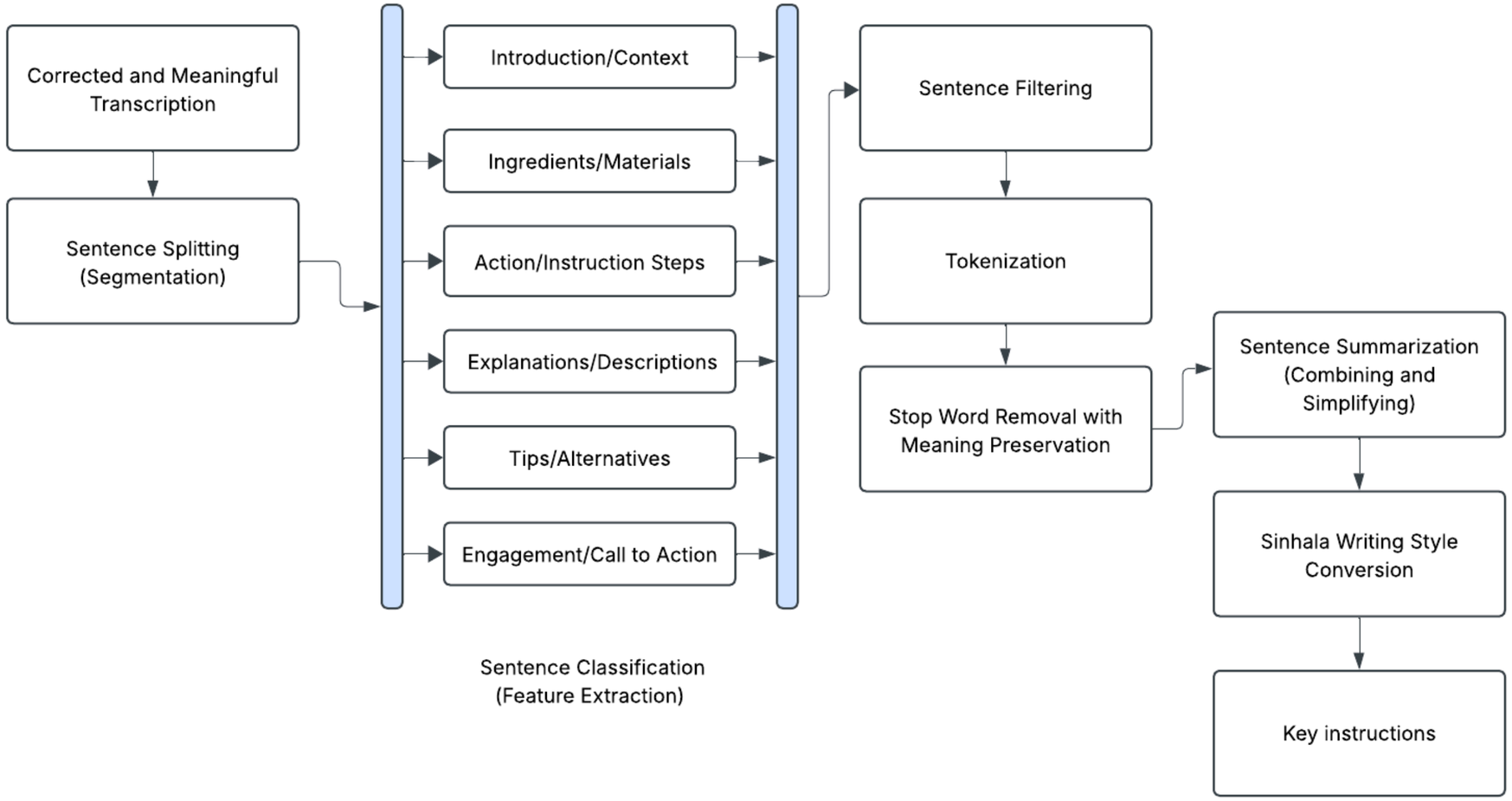
Introduction

- The section focuses on identifying the key instructions from the transcribed text. Afterwards, identified key instructions are written in instruction format.

Challenges

- Instructions do not always follow a fixed structure in the transcribed text.
- Limited NLP resources for Sinhala language.
- Some words in Sinhala have multiple meanings depending on the context, which can cause misinterpretation.

Analysis & Design



Methodology

- **Sentence Splitting (Segmentation)/Tokenization** : Indic NLP Library

- **Sentence Classification** :
  - Create a labeled dataset
    - Labels: INTRO, INGREDIENT, ACTION, DESCRIPTION, TIP, or ENGAGEMENT
  - Use pre-trained multilingual transformer models to generate sentence embeddings that capture semantic meaning.
  - Fine-tune a classification model on these embeddings to accurately assign labels.
  - Filter out non-essential sentences and retain only the relevant ones.

- **Stop Word Removal with Meaning Preservation** :
  - Use a custom Sinhala stop word list tailored for cooking instructions.
  - Dependency parsing : Ensure that stop word removal does not unintentionally delete words that play a key role in sentence meaning.

Methodology

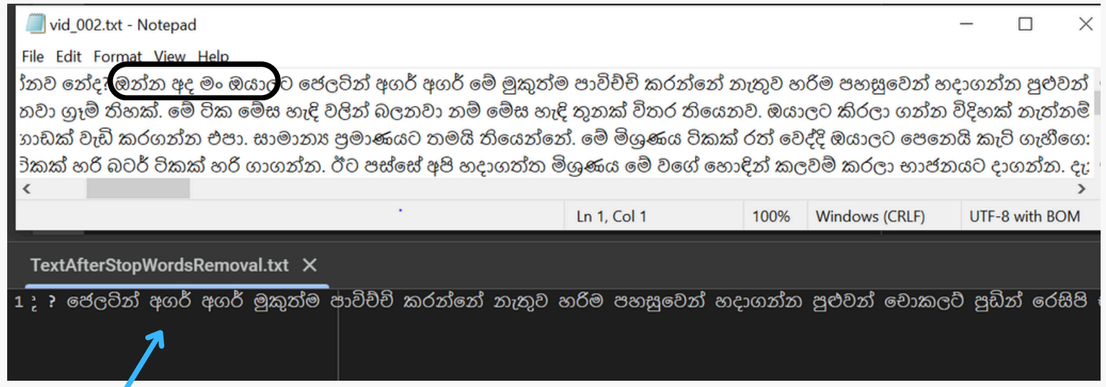
- **Sentence Summarization (Combining and Simplifying)** :
  - Rule-based summarization technique to combine and simplify multiple related sentences.
  - Use NLP techniques such as Tokenization, Dependency parsing, Regular expressions.

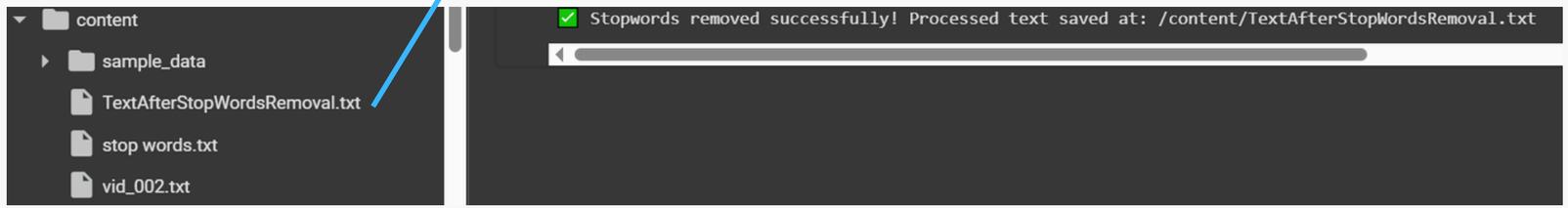
- **Sinhala Writing Style Conversion** :
  - Use Sequence-to-Sequence Model to convert informal Sinhala (spoken style) to formal Sinhala (written style).

Implementation

Tokenization, Sentence Segmentation and Stop word Removal

```
!pip install indic-nlp-library

from indicnlp.tokenize import sentence_tokenize, indic_tokenize

# Read stopwords from the file
stopwords_file = "/content/stop words.txt"  # Change this to the correct path
with open(stopwords_file, "r", encoding="utf-8") as f:
    stop_words = set(f.read().splitlines())  # Store stopwords as a set for faster lookup

# Read input text
input_file = "/content/vid_002.txt"
with open(input_file, "r", encoding="utf-8") as f:
    text = f.read()


def remove_stopwords(text):
    sentences = sentence_tokenize.sentence_split(text, "si")  # Sentence Splitting
    tokenized_sentences = [indic_tokenize.trivial_tokenize(sent) for sent in sentences]  # Tokenization
    # Stopword Removal
    filtered_sentences = [[word for word in tokens if word not in stop_words] for tokens in tokenized_sentences]
    # Reconstruct sentences
    reconstructed_text = [" ".join(tokens) for tokens in filtered_sentences]
    return reconstructed_text

Final_output = remove_stopwords(text)

output_file = "/content/TextAfterStopWordsRemoval.txt"
with open(output_file, "w", encoding="utf-8") as f:
    f.write(" ".join(Final_output))

print(f"✅ Stopwords removed successfully! Processed text saved at: {output_file}")
```





Module 3 : Action/Object Detection and Image Selection Aligned with Instructions.

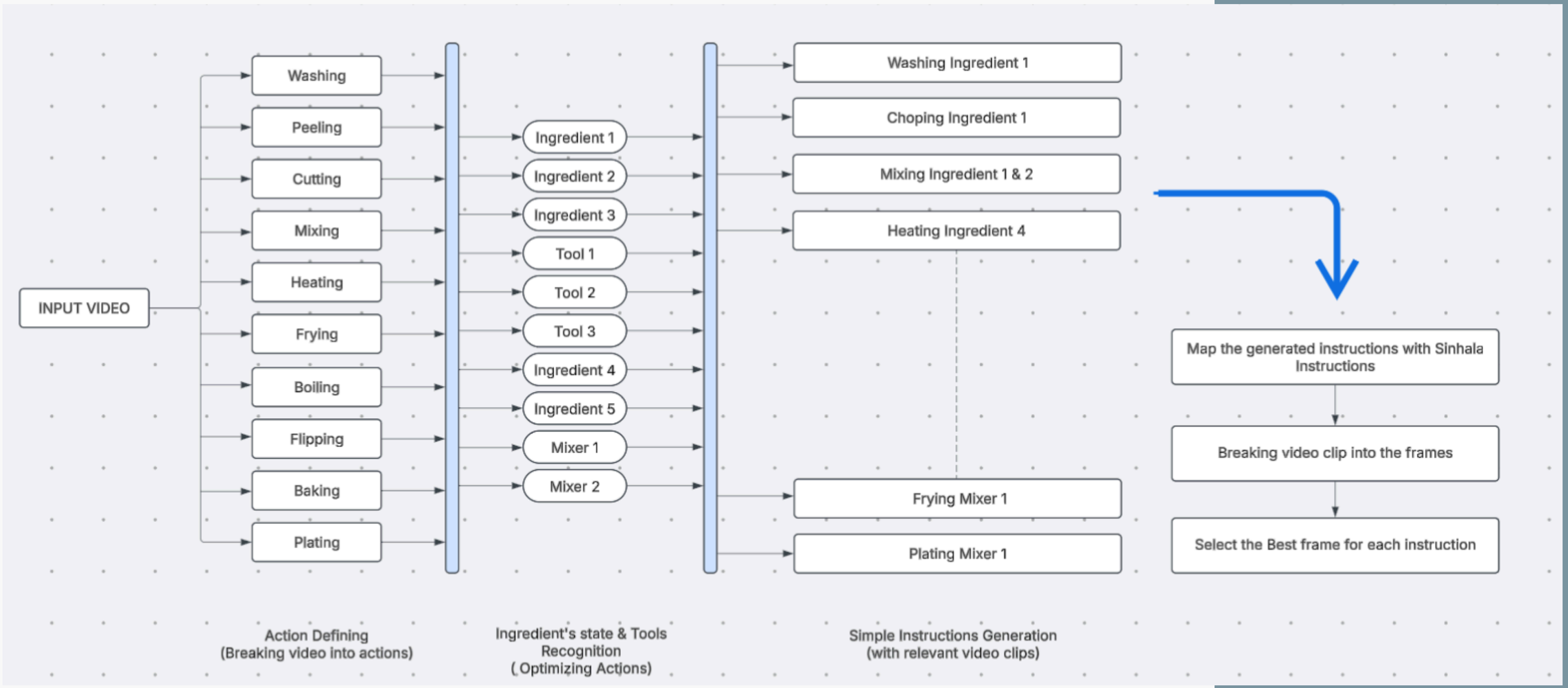
Introduction

- Focuses on defining cooking actions, identifying objects and tools, generating step-by-step instructions, mapping them with Sinhala translations, and selecting relevant video frames

Challenges

- Action defining, Text Mapping, image selection, and performance guide quality using machine learning techniques.
- Evaluate and benchmark the success factor for the algorithms used for the above operations.

Module 3 Diagram



Example

Step	Define Action	Recognized Object	Generated Instruction	Sinhala Instruction	Selected Frame
1	Cracking	Egg	Crack the egg into a bowl.	බිත්තරය බෙදී එකට කඩිය යුතුය.	
2	Chopping	Knife, Onion	Chop the onion into pieces.	ලූනු කුඩාකෙඩස් වශයෙන් කපා ගන්න.	
3	Mixing	Pan, Spoon, Egg, Onion	Mix the egg and onion well.	බිත්තරය සහ ලූනු හොඳින් මිශ්‍ර කරන්න.	
4	Frying	Egg Mixture, Pan	Pour the mixture into the pan and fry.	මිශ්‍රණය පඳන එකට දමා කර කර ගන්න.	

Methodology

Action Defining and Optimizing

- Object Detection Models: YOLO (You Only Look Once)
- Libraries: OpenCV, TensorFlow, PyTorch
- Frameworks: MMAction2 (for basic human action recognition)

Instruction Generations

- Natural Language Processing (NLP): GPT, BERT, T5
- Libraries: NLTK, SpaCy, Transformers

Mapping with Sinhala instructions

- Translation APIs: Google Translate API, MarianMT

Frame Selection

- Frame Selection Models: CNN-based Image Classification, Keyframe Extraction
- Libraries: OpenCV, Scikit-Image, FFmpeg

Implementation

Actions Defining and Optimizing

ActionRecognition main

ActionRecognition.py

```python
# Set paths
video_path = "VID_006.mp4"  # Original video
output_frame_folder = "output_frames"  # Folder to save frames
output_video_folder = "output_videos"  # Folder to save video clips

# Create output directories if they don't exist
os.makedirs(output_frame_folder, exist_ok=True)
os.makedirs(output_video_folder, exist_ok=True)

# Open video file
cap = cv2.VideoCapture(video_path)
fps = int(cap.get(cv2.CAP_PROP_FPS))  # Get FPS
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

Run ActionRecognition

```
Generating video: output_videos\scene_6.mp4 from frames 1558 to 2267
Generating video: output_videos\scene_7.mp4 from frames 2322 to 2773
Generating video: output_videos\scene_8.mp4 from frames 2842 to 3189
Generating video: output_videos\scene_9.mp4 from frames 3222 to 3250
Generating video: output_videos\scene_10.mp4 from frames 4093 to 4128
Generating video: output_videos\scene_11.mp4 from frames 4179 to 4214
Generating video: output_videos\scene_12.mp4 from frames 4298 to 4298
Generating video: output_videos\scene_13.mp4 from frames 4365 to 4368
Generating video: output_videos\scene_14.mp4 from frames 4821 to 4824
Generating video: output_videos\scene_15.mp4 from frames 5551 to 5551
Generating video: output_videos\scene_16.mp4 from frames 6039 to 6040
Generating video: output_videos\scene_17.mp4 from frames 7234 to 7234
Generating video: output_videos\scene_18.mp4 from frames 7861 to 7861
```

ActionRecognition > ActionRecognition.py

Thank you