Enhanced Media Experience with AI-Powered Commercial Detection and Replacement - Aryan Nanda

Table of contents

Chapter 1

Introduction

The BeagleBone® AI-64 from the BeagleBoard.org Foundation is a complete system for developing artificial intelligence (AI) and machine-learning solutions with the convenience and expandability of the BeagleBone platform and onboard peripherals to start learning and building applications. Leveraging the capabilities of BeagleBoard's powerful processing units, the project will focus on creating a real-time, efficient solution that enhances media consumption experiences by seamlessly integrating custom audio streams during commercial breaks.

1.1 Summary links

- **Contributor:** Aryan Nanda
- **Mentors:** Jason Kridner, Deepak Khatri
- **GSoC Repository:** TBD

1.2 Status

This project is currently just a proposal.

1.3 Proposal

- Created accounts across OpenBeagle, Discord and Beagle Forum
- The PR Request for Cross Compilation: #185
- Created a project proposal using the proposed template

1.4 About

- **Resume** - Find my resume here
- **Forum:** u/aryan_nanda
- **OpenBeagle:** aryan_nanda
- **Github:** AryanNanda17
- **School:** Veermata Jijabai Technological Institute (VJTI)
- **Country:** India

- **Primary language:** English, Hindi

- **Typical work hours:** 9AM-5PM Indian Standard Time

- **Previous GSoC participation:** This would be my first time participating in GSOC

Chapter 2

About the Project

Project name: Enhanced Media Experience with AI-Powered Commercial Detection and Replacement

2.1   Description

I propose developing **GStreamer Plugins** capable of processing video inputs based on their classification. The plugins will identify commercials and either replace them with alternative content or obscure them, while also substituting the audio with predefined streams. This enhancement aims to improve the media consumption experience by eliminating unnecessary interruptions. I intend to **explore various video classification models** to achieve accurate detection and utilize TensorFlow Lite to leverage the **native accelerators of BeagleBone AI-64** for high-performance, real-time inferencing with minimal latency. I believe real-time high-performance would be the most critical thing for this project and I intend on testing a few different ways to see which one works best.

2.2   Goals and Objectives

The goal of this project is to detect and replace commercials in video streams on BeagleBoard hardware using a GStreamer pipeline which includes a model that accurately detects commercials with minimal latency. Comparison of different model accuracy can be done by doing some manual analysis and trying different video classification models and to finally use the best performing option to be included in the GStreamer pipeline for inferencing of real-time videos. This would be the result presented at the end of the project timeline. For phase 1 evaluation, the goal is to build a training dataset, preprocess it and fine-tune and train a Video Classification model. For phase 2 evaluation, the goal is to use the the best model identified in phase 1 for commercial detection and build a GStreamer pipeline and use native accelerators present in BeagleBone AI-64 for high-performance.

In order to accomplish this project the following objectives need to be met.

1. **Phase 1:-**

   - Develop a dataset of videos and corresponding labels indicating the presence of commercials in specific segments.

   - Preprocess the dataset to ensure it's suitable for input into deep learning models. Moreover divide the datset into train, validation and test set.

   - Apply transfer learning and Fine-tune various deep learning models and train them on the prepared dataset to identify the most accurate one for commercial detection in videos.

   - Save all trained models to local disk and perform real-time inference using OpenCV to determine the model that yields the best results with high-performance.

2. **Phase 2:-**

- Based on all the options tried in Phase 1, decide on the final model to be used in the GStreamer pipeline.

- Compiling the model and generating artifacts so that we can use it in TFLite Runtime.

- Building a GStreamer pipeline that would take real-time input of media and would identify the commercial segments in it.

- If the commercial segment is identified the GStreamer pipeline would either replace them with alternative content or obscure them, while also substituting the audio with predefined streams.

- I will also try to cut the commercial out completely and splice the ends.

- Enhancing the Real-time performance using native hardware Accelerators present in Beagle-Bone AI-64.

Chapter 3

Methods

In this section, I will individually specify the training dataset, model, GStreamer Pipeline etc. methods that I plan on using in greater details.

3.1 Building training Dataset and Preprocessing

To train the model effectively, we need a dataset with accurate labels. Since a suitable commercial video dataset isn't readily available, I'll create one. This dataset will consist of two classes: commercial and non-commercial. By dividing the dataset into Commercial and Non-Commercial segments, I am focusing more on "Content Categorization". Separating the dataset into commercials and non-commercials allows our model to learn distinct features associated with each category. For commercials, this might include fast-paced editing, product logos, specific jingles, or other visual/audio cues. Non-commercial segments may include slower-paced scenes, dialogue, or narrative content.

To build this dataset, I'll refer to the **Youtube-8M dataset** [1], which includes videos categorized as TV advertisements. However, since the Youtube-8M dataset provides encoded feature vectors instead of the actual videos, direct usage would result in significant latency. Therefore, I'll use it as a reference and download the videos labeled by it as advertisements to build our dataset. I will use web scraper to automate this process by extracting URLs of the commercial videos. For the non-commercial part, I will download random videos from other categories of Youtube-8M dataset. After the dataset is ready I will preprocess it to ensure it's suitable for input into deep learning models.

Moreover I'll divide the datset into train, validation and test set. To address temporal dependencies during training, I intend to employ random shuffling of the dataset using `tf.keras.preprocessing. image_dataset_from_directory() with shuffle=True`. This approach ensures that videos from different folders are presented to the model randomly, allowing it to learn scene change detection effectively.

3.2 Video Classification models

MoViNets is a good model for our task as it can operate on streaming videos for online inference. The main reason behind trying out MoViNets first is becaue it does quick and continuous analysis of incoming video streams. MoViNet utilizes NAS(Neural Architecture Search) to balance accuracy and efficiency, incorporates stream buffers for constant memory usage [Fig. 1], and improves accuracy via temporal ensembles [2]. The MoViNet architecture uses 3D convolutions that are "causal". Causal convolution ensures that the output at time t is computed using only inputs up to time t [2][Fig. 2]. This allows for efficient streaming. This make MoViNets a perfect choice for our case.

Since we don't have a big dataset, we will use the pre-trained MoViNets model as a feature extractor and fine-tune it on our dataset. I will remove the classification layers of MoViNets and use its pre-trained weights to extract features from our dataset. Then, train a smaller classifier (e.g., a few fully connected layers) on top

of these features. This way we can use the features learned by MoViNets on the larger dataset with minimal risk of overfitting. This can help improve the model's performance even with limited data.
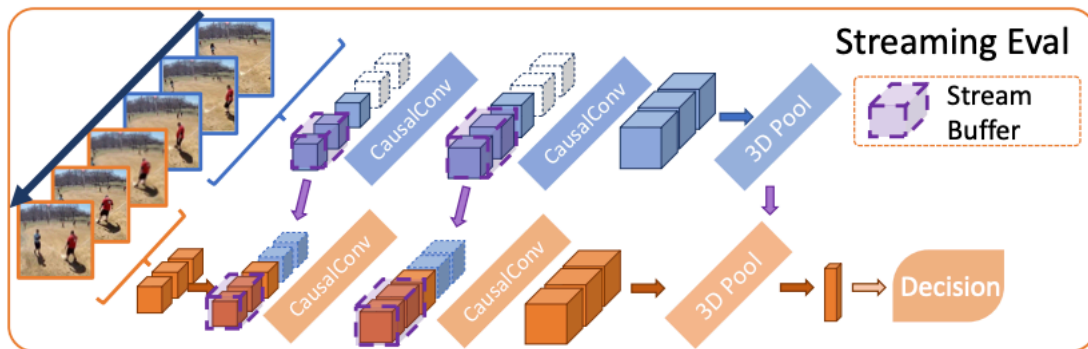

Figure 1: Stream buffer in MoViNets [2]
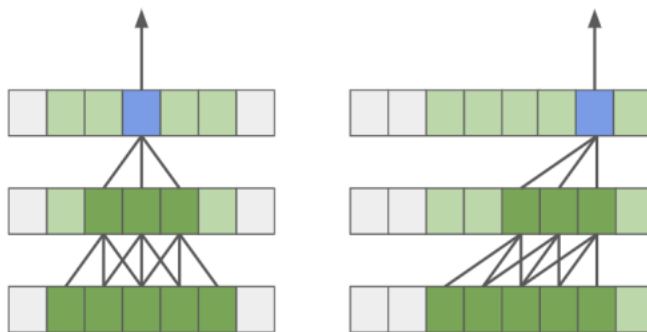
Figure 2: Standard Convolution Vs Causal Convolution [2]

If MoViNet does not perform well than we can use other models like **Conv+LSTMs** [Fig. 3][3]. Since a video is just a series of frames, a naive video classification method would be pass each frame from a video file through a CNN, classify each frame individually and independently of each other, choose the label with the largest corresponding probability, label the frame, and assign the most assigned image label to the video.

To solve the problem of "prediction flickering", where the label for the video changes rapidly when scenes get labeled differently. I will use **rolling prediction averaging** to reduce "flickering" in results. And I will maintain a queue to store the last few frames and whenever a scene change is detected, all frames in the queue would be marked with the current result, allowing for retroactive scene modification. The depth of the queue will be determined through experimentation to find the optimal setting.

The Conv+LSTMs model will perform well as it considers both the spatial and temporal features of videos just like a Conv3D model. The only reason it is not my first choice is because MoViNets are considered to be better for real-time performance.

Figure 3: Conv+LSTMs [3]

3.2.1 Optional Methods

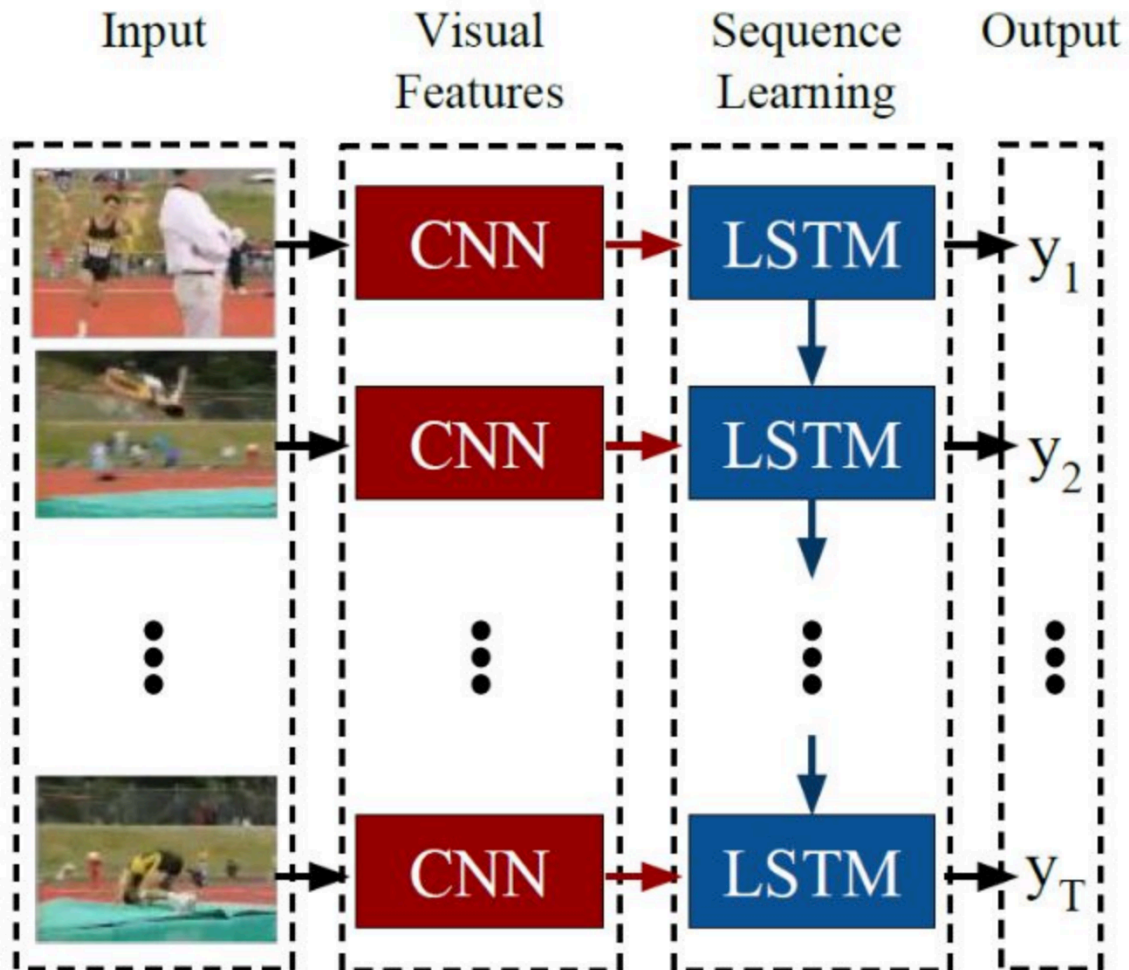- **ViViT**: A Video Vision Transformer, this is a pure Transformer based model which extracts spatio-temporal tokens from the input video, which are then encoded by a series of transformer layers [4]. I have kept this as an optional method because our problem is of binary classification(either Commercial or Non-Commercial), so using such a complex model for this small problem may not be as efficient as other models.

- **Audio fingerprinting**: This method involves extracting unique characteristics or features from audio signals to create a compact representation, often called a fingerprint. These fingerprints can then be compared against a database or used for various audio processing tasks [5]. I have kept it as an optional method because it may sometimes yield poorer results compared to deep learning models like MoViNets and Conv+LSTMs, particularly in tasks requiring complex audio understanding.

- **Scene Change Detection**: This approach involve detecting scene changes to segment the video into distinct segments or shots based on difference between pixel values of two consecutive frames etc [6]. And then applying video classification model on segmented frames. I have kept this as an optional appraoch because I think adding an additional step would cause unnecessary challenges.

3.3 Choosing the Best Performing model

I will choose the best performing model based on model's performance on:-

1. **Evaluation metrics**

   - Accuracy: This metric provides a general overview of how well our model is performing overall. It's a good starting point for evaluating performance, but it might not be sufficient on its own, especially if the classes are imbalanced.

   - Precision and Recall: These metrics provide insights into the model's ability to minimize false positives (precision) and false negatives (recall). Since our problem involves binary classification, precision and recall are essential for understanding the model's performance on each class (commercial and non-commercial).

   - F1 Score: It provides a balanced measure of the model's performance. A high F1 score indicates that the model effectively balances precision and recall, ensuring both high accuracy and coverage in detecting commercials.

2. **Real-time inferencing**

   - I will use OpenCV to evaluate real-time performance of the models. This ensures that our model's performance is evaluated under conditions similar to those it will encounter in deployment.

This code snippet illustrates how the model will be assessed in real-time, focusing on both **detection accuracy** and **frames per second (FPS)** as the primary evaluation metrics.

```python
# Import necessary libraries
from keras.models import load_model
from collections import deque
import numpy as np
import pickle
import cv2
import time

model = load_model("./our_trained_commercial_detection_model")      # Load
↪the pre-trained Keras model
lb = pickle.load(open("./videoClassification.pickle", "rb"))        # Load
↪the label binarizer used during training
mean = np.array([123.68, 116.779, 103.939], dtype="float32")        # Define
↪the mean value for image preprocessing
Queue = deque(maxlen=128)                                           # Define
↪a deque (double-ended queue) to store predictions of past frames
capture_video = cv2.VideoCapture("./example_clips/DemoVideo.mp4")   # Open
↪the video file for reading
(Width, Height) = (None, None)                                      #
↪Initialize variables for storing frame dimensions and previous time
ptime = 0

# Loop through each frame of the video
while True:
    (taken, frame) = capture_video.read()                           # Read a
↪frame from the video

    if not taken:                                                   # Break
↪the loop if no more frames are available
        break

    if Width is None or Height is None:                             # Get
↪frame dimensions if not already obtained
        (Width, Height) = frame.shape[:2]

    output = frame.copy()                                           # Make a
↪copy of the frame for processing and classification

    #--------------------------Pre-Processing-----------------------#
```

(continues on next page)

```python
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)                     #
→Preprocess the frame: convert color space and resize
    frame = cv2.resize(frame, (sizeRequiredByModelHere)).astype("float32")
    frame -= mean                                                      #
→Subtract the mean value for normalization


    #-------------------------model-inferencing---------------------#

    preds = model.predict(np.expand_dims(frame, axis=0))[0]          # Make
→predictions on the preprocessed frame
    Queue.append(preds)                                              # Append
→the predictions to the deque
    results = np.array(Queue).mean(axis=0)                           #
→Calculate the average of predictions from past frames
    i = np.argmax(results)                                           # Get
→the index of the class with the highest average prediction
    label = lb.classes_[i]                                           # Get
→the label corresponding to the predicted class


    #-------------------------Post-Processing-------------------------#

    if label == "commercial":                                        # Apply
→Gaussian blur to the output frame if the label is "commercial"
        output = cv2.GaussianBlur(output, (99, 99), 0)

    ctime = time.time()                                              #
→Calculate and display FPS
    fps = int(1 / (ctime - ptime))
    cv2.putText(output, str(fps), (20, 200), cv2.FONT_HERSHEY_SIMPLEX, 2,
→(255, 0, 255), 2)
    ptime = ctime

    cv2.imshow("In progress", output)                                # Show
→the processed frame

    key = cv2.waitKey(1) & 0xFF                                      # Handle
→user input (press 'q' to quit)
    if key == ord("q"):
        break

# Release resources when finished
capture_video.release()
cv2.destroyAllWindows()
```

3.4 Model Execution on BeagleBone AI-64

BeagleBone AI-64 Linux for Edge AI supports importing pre-trained custom models to run inference on target. Moreover, Edge AI BeagleBone AI-64 images have TensorFlow Lite already installed with acceleration enabled. The Debian-based SDK makes use of pre-compiled DNN (Deep Neural Network) models and performs inference using various OSRT (open source runtime) such as TFLite runtime, ONNX runtime etc.

In order to infer a DNN, SDK expects the DNN and associated artifacts in the below directory structure [7].

```
project_root
|
├── param.yaml
|
├── artifacts
|    ├── 264_tidl_io_1.bin
```

```
|   ├── 264_tidl_net.bin
|   ├── 264_tidl_net.bin.layer_info.txt
|   ├── 264_tidl_net.bin_netLog.txt
|   ├── 264_tidl_net.bin.svg
|   ├── allowedNode.txt
|   └── runtimes_visualization.svg
|
└── model
    └── my_commercial_detection_model.tflite
```

1. model: This directory contains the DNN being targeted to infer.

2. artifacts: This directory contains the artifacts generated after the compilation of DNN for SDK.

3. param.yaml: A configuration file in yaml format to provide basic information about DNN, and associated pre and post processing parameters.

Therefore, after choosing the model to be used in GStreamer pipeline, I will generate the artifacts directory by following the instructions mentioned in TexasInstruments:edgeai-tidl-tools examples [7].
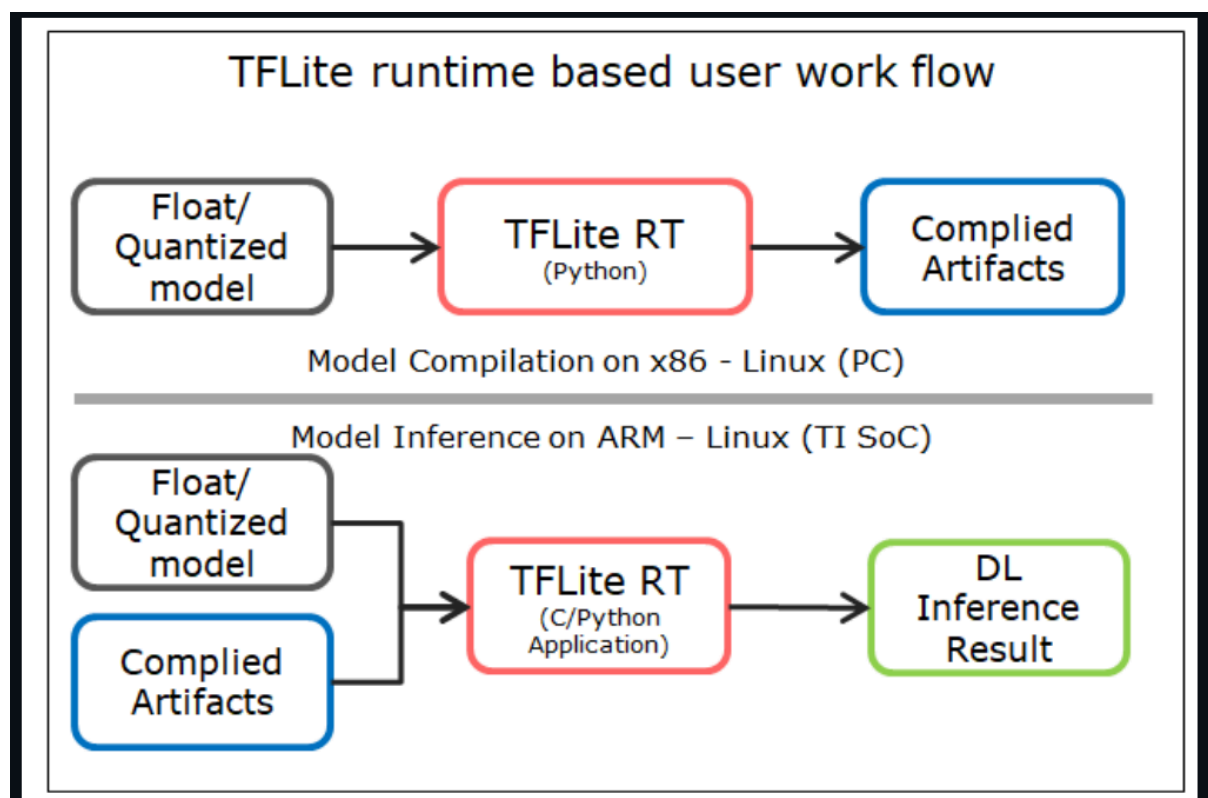

Figure 4: TFLite Runtime [7]

3.5   GStreamer Pipeline

The data flow in the GStreamer pipeline at a high level can be split into 3-parts [8]:-

1. Input Pipeline - Grabs a frame from the input source.

2. Output Pipeline - Sends the output to the display.

3. Compute Pipeline - Performs pre-processing, inference and post-processing.

I will create a GStreamer Pipeline that will receive input from an **hdmi source** and it will grab it frame by frame. The frame will be split into two paths.

The "analytics" path normalizes the frame and resizes the input to match the resolution required to run the deep learning model. The "visualization" path is provided to the post-processing module which does the required post process required by the model. If a commercial video is detected, we apply blurring to the video frames and replace the audio. If a non-commercial video is detected, proceed with the normal visualization process without blurring or replacing the audio. Post-processed output is then sent to display [8].

NNStreamer provides efficient and flexible data streaming for machine learning applications, making it suitable for tasks such as running inference on video frames. So, I will use NNStreamer elements to do inferencing of videos.
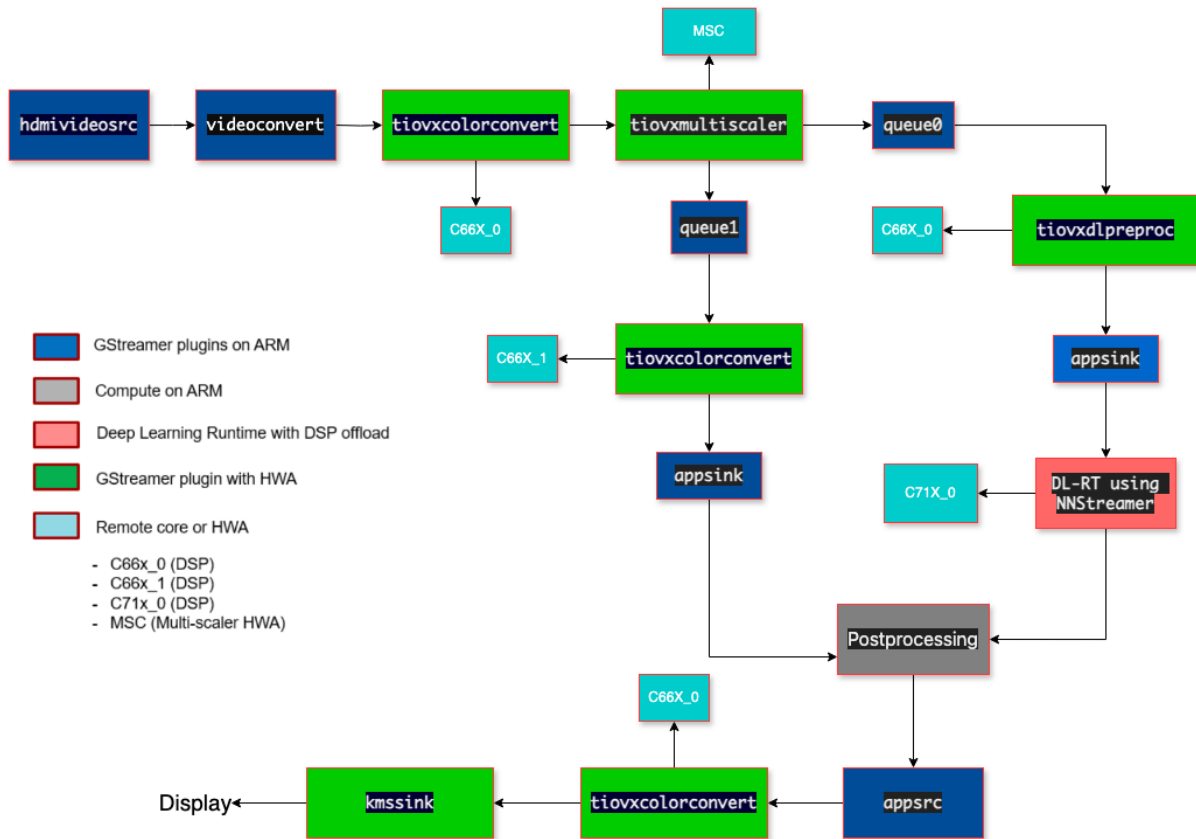

Figure 5: GStreamer Pipeline [8]

The above GStreamer pipeline is a demo pipeline inspired from edge_ai_apps/data_flows [8] and there could be a few more changes to it depending upon our specific need.

- **"hdmisrc"** element is used for capturing audio and video data from an HDMI source.

- **"videoconvert"** ensure proper format conversion for display.

- **"tiovxcolorconvert"** is used to perform color space conversion.

- **"tiovxmultiscaler"** is used to perform multi-scaling operations on video frames. It allows us to efficiently scale the input frames to multiple desired resolutions in a single step.

- **"tiovxdlpreproc"** is used to perform pre-processing of input data in deep learning inference pipelines using TIOVX (TI OpenVX) framework.

- **"kmssink"** is used for displaying video on systems.
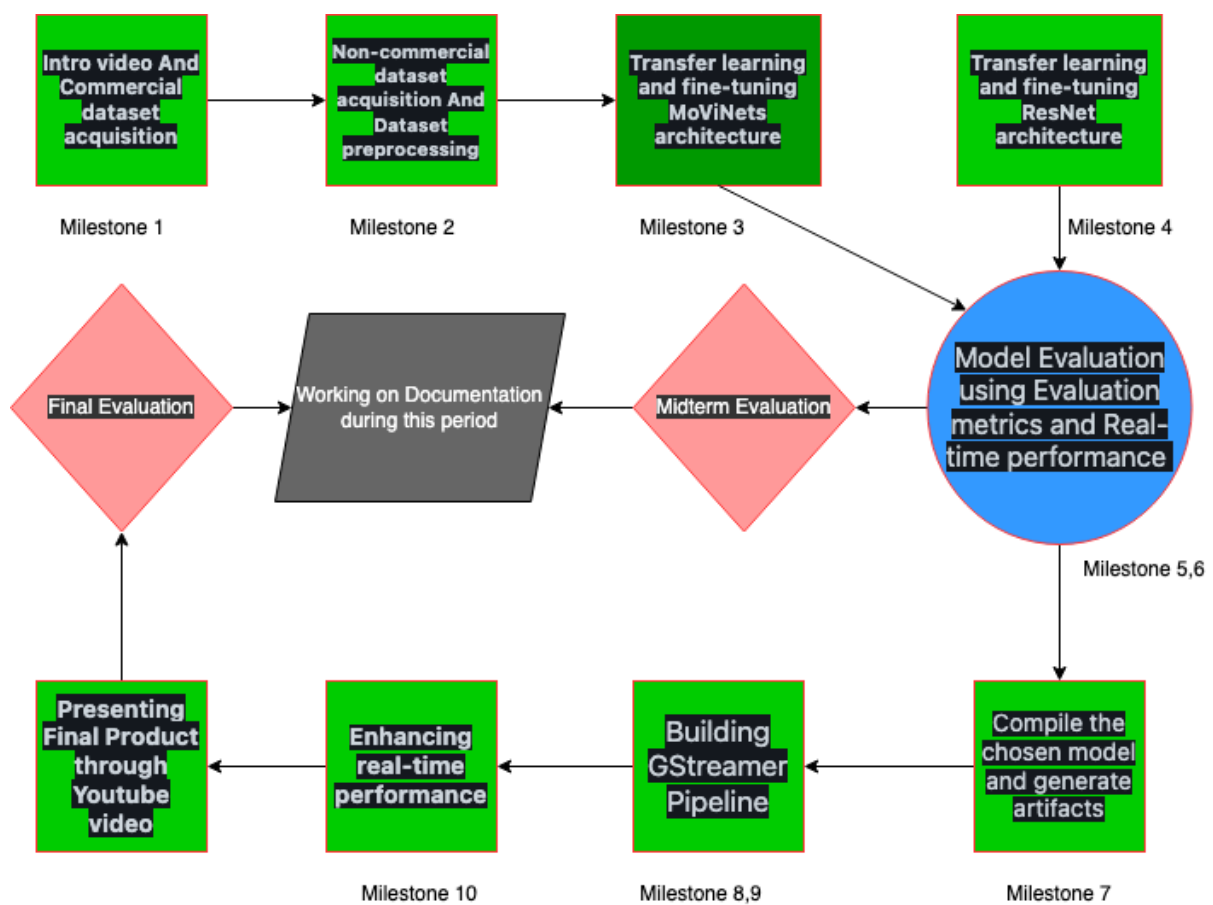
3.6   Project Workflow

Figure 6: Project Workflow

3.7   Software

- Python
- C++
- TensorFlow
- TFLite
- GStreamer
- OpenCV
- Build Systems

3.8   Hardware

- Ability to capture and display video streams using BeagleBone AI-64

Chapter 4

Timeline

4.1 Timeline summary

Date	Activity
February 26 - March 3	Connect with possible mentors and request review on first draft
March 4 - March 10	Complete prerequisites, verify value to community and request review on second draft
March 11 - March 20	Finalized timeline and request review on final draft
March 21 - April 2	Proposal review and Submit application
April 3 - May 1	Understanding GStreamer pipeline and TFLite runtime of BeagleBone AI-64.
May 2 - May 10	*Community Bonding Period (May 1st - May 10th)*
May 11 - May 31	Focus on college exams.
June 1 - June 3	Start coding and introductory video
June 3 - June 9	*Milestone #1, Releasing introductory video and developing commercial dataset (June 3)*
June 10 - June 16	*Milestone #2, Developing non-commercial dataset and dataset preprocessing (June 10)*
June 17 - June 23	*Milestone #3, Transfer learning and fine-tuning MoViNets architecture (June 17)*
June 24 - June 30	*Milestone #4, Transfer learning and fine-tuning ResNet architecture (June 24)*
July 1 - July 7	*Milestone #5, Evaluate performance metrics to choose the best-performing model (July 1)*
July 8 - July 14	*Submit midterm evaluations (July 8th)*
July 15 - July 21	*Milestone #6, Finalizing the best model by performing real-time inferencing (July 15)*
July 22 - July 28	*Milestone #7, Compiling the model and generating artifacts and building pre-processing part of GStreamer pipeline (July 22)*
July 29 - August 4	*Milestone #8, Building the compute pipeline using NNStreamer (July 29)*
August 5 - August 11	*Milestone #9, Building the post-processing part of GStreamer pipeline (August 5)*
August 12 - August 18	*Milestone #10, Enhancing real-time performance (August 12)*
August 19	*Submit final project video, submit final work to GSoC site and complete final mentor evaluation (August 19)*

4.2 Timeline detailed

4.2.1 Community Bonding Period (May 1st - May 10th)

- Discuss implementation ideas with mentors.
- Discuss the scope of the project.

4.2.2 Milestone #1, Releasing introductory video and developing commercial dataset (June 3)

- Making an Introductory Video.
- **Commercial dataset acquisition:**

- – Web scrape videos marked as advertisements from YouTube 8-M dataset.
- – Ensure proper labeling and categorization of commercial videos.

4.2.3 Milestone #2, Developing non-commercial dataset and dataset preprocessing (June 10)

- **Non-commercial dataset acquisition:**
  - – Web scrape random videos from other categories of YouTube 8-M dataset.
  - – Ensure diversity and relevance of non-commercial videos.
- **Dataset preprocessing:**
  - – Preprocess acquired datasets for suitability in deep learning models.
  - – Divide datasets into train, validation, and test sets.
  - – Perform random shuffling of data to maintain temporal dependencies.

4.2.4 Milestone #3, Transfer learning and fine-tuning MoViNets architecture (June 17)

- **Transfer learning and fine-tuning MoViNets architecture:**
  - – Apply transfer learning on MoViNets and fine-tune its last few layers.
  - – Train MoViNets on the prepared dataset for video classification.

4.2.5 Milestone #4, Transfer learning and fine-tuning ResNet architecture (June 24)

- **Transfer learning and fine-tuning ResNet architecture:**
  - – Adding additional layers of LSTMs for extracting temporal dependencies.
  - – Developing ResNet-LSTMs model architecture for video classification.
  - – Train the ResNet-LSTMs model on the prepared dataset.

4.2.6 Milestone #5, Evaluate performance metrics to choose the best-performing model (July 1)

- **Finalize the best model:**
  - – Save all trained models to local disk
  - – Evaluate performance metrics to choose the best-performing model.

4.2.7 Submit midterm evaluations (July 8th)

- Document the progress made during the first phase of the project.

Important: **July 12 - 18:00 UTC:** Midterm evaluation deadline (standard coding period)

4.2.8 Milestone #6, Finalizing the best model by performing real-time inferencing (July 15)

- **Finalize the best model:**
  - Perform real-time inference using OpenCV to determine the model that yields the best results with high-performance.
  - Based on all the options tried in Phase 1, decide on the final model to be used in the GStreamer pipeline.

4.2.9 Milestone #7, Compiling the model and generating artifacts and building pre-processing part of GStreamer pipeline (July 22)

- Compile the chosen model and generate artifacts for TFLite runtime.
- **Building the pre-processing part of GStreamer pipeline:**
  - Develop the pre-processing module to prepare video frames for inference.

4.2.10 Milestone #8, Building the compute pipeline using NNStreamer (July 29)

- **Building the compute pipeline using NNStreamer:**
  - Implement NNStreamer for inferencing videos using the compiled model.

4.2.11 Milestone #9, Building the post-processing part of GStreamer pipeline (August 5)

- **Building the post-processing part of GStreamer pipeline:**
  - Develop the post-processing module to perform actions based on classification results.
  - Implement replacement or obscuring of commercial segments and audio substitution.

4.2.12 Milestone #10, Enhancing real-time performance (August 12)

- **Enhancing real-time performance:**
  - Optimize the GStreamer pipeline for real-time performance using native hardware accelerators.
  - Ensure smooth and efficient processing of video streams.

4.2.13 Submit final project video, submit final work to GSoC site and complete final mentor evaluation (August 19)

- Submit final project video, submit final work to GSoC site and complete final mentor evaluation.

4.2.14 Final Submission (Aug 24nd)

Important: August 19 - 26 - 18:00 UTC: Final week: GSoC contributors submit their final work product and their final mentor evaluation (standard coding period)

August 26 - September 2 - 18:00 UTC: Mentors submit final GSoC contributor evaluations (standard coding period)

4.2.15   Initial results (September 3)

Important: **September 3 - November 4:** GSoC contributors with extended timelines continue coding

November 4 - 18:00 UTC: Final date for all GSoC contributors to submit their final work product and final evaluation

November 11 - 18:00 UTC: Final date for mentors to submit evaluations for GSoC contributor projects with extended deadline

Chapter 5

Experience and approach

This project requires prior experience with machine learning, multimedia processing and embedded systems.

- As a good starting point for this project, I build a Sports Video Classification model and did **Video Processing on it based on Video classification** using OpenCV(Video Processing based on Video classification is an important part of the project). Demo

- **We will be building Pure C++ GStreamer pipeline from input to output so experience with C++ Codebases and build systems is required.**

    – Relevant contribution - #123 (OpenCV/C++)

- I have Previously Worked on the project GestureSense in which I did Image Processing based on Image classification using OpenCV/Python.

- I have past experience with esp-32 microcontroller and I have Previously Worked on a project Multi-Code-Esp in which I build a multi-code esp component.

- **Experience in Open-Source**

    Contributed at pymc repository. Added enhancements.

    – #7132 (merged)

    – #7125 (merged)

    Resolved one issue in OpenCV repository (Improved Documentation).

    – #22177 (merged)

- **Contributions in openbeagle.org/gsoc**

    – Resolved pdf pageBreak issue - #33 (merged)

    – Added New idea - #25 (merged)

    – Improved Documentation - #23 (merged)

5.1 Contingency

- **If I get stuck on my project and my mentor isn't around, I will use the following resources:-**

    – MoViNets

    – GStreamer Docs

    – BeagleBone AI-64

    – NNStreamer

- Moreover, the BeagleBoard community is extremely helpful and active in resolving doubts, which makes it a great going for the project resources and clarification.

- I intend to remain involved and provide ongoing support for this project beyond the duration of the GSOC timeline.

5.2 Benefit

This project will not only enhance the media consumption experience for users of BeagleBoard hardware but also serve as an educational resource on integrating AI and machine learning capabilities into embedded systems. It will provide valuable insights into:

- The practical challenges of deploying neural network models in resource-constrained environments.

- The development of custom GStreamer plugins for multimedia processing.

- Real-world applications of machine learning in enhancing digital media experiences.

5.3 Misc

- The PR Request for Cross Compilation: #185

- Relevant Coursework: Neural Networks and Deep Learning, Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization, Convolutional Neural Networks

Chapter 6

References

1. Youtube: YouTube-8M: A Large and Diverse Labeled Video Dataset

2. Dan Kondratyuk*, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, Boqing Gong: MoViNets: Mobile Video Networks for Efficient Video Recognition.

3. Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell: Long-term Recurrent Convolutional Networks for Visual Recognition and Description

4. Anurag Arnab* Mostafa Dehghani* Georg Heigold Chen Sun Mario Lucič c´† Cordelia Schmid†: ViViT: A Video Vision Transformer

5. Nilesh M. Patil, Dr. Milind U. Nemade: Content-Based Audio Classification and Retrieval: A Novel Approach

6. Igor Bieda, Anton Kisil, Taras Panchenko: An Approach to Scene Change Detection

7. TexasInstruments: edgeai-tidl-tools

8. BeagleBone AI-64: data-flows in edge_ai_apps